AD-A066 722 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB N Y F/G 9/2
A DATA BASE MANAGEMENT MODELING TECHNIQUE AND SPECIAL FUNCTION --ETC(U)
JAN 79 G T CAPRARO, P B BERRA

UNCLASSIFIED RADC-TR-79-14 NL

1 OF 3
ADA
066722

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-14
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A DATA BASE MANAGEMENT MODELING TECHNIQUE AND SPECIAL FUNCTION HARDWARE ARCHITECTURE
5. TYPE OF REPORT & PERIOD COVERED: In-House Report
6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): Gerard T. Capraro and P. Bruce Berra (Syracuse University)
8. CONTRACT OR GRANT NUMBER(s): N/A

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Rome Air Development Center (RBCT), Griffiss AFB NY 13441
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F; 23380320; 17 03

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (RBCT), Griffiss AFB NY 13441
12. REPORT DATE: January 1979
13. NUMBER OF PAGES: 248

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Dr. Gerard T. Capraro

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Architecture
Data Base Management
Set Theory
Electromagnetic Compatibility

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research is concerned with the development of a mathematical base that can be utilized to model data base management systems from the user level down to the bit level and to develop and evaluate proposed hardware that could be utilized to implement a data dictionary and part of a data directory.

The mathematical modeling development is accomplished through set theory and the addition of order to sets. This mathematical base is used to define in detail some of the functions that must be performed in Data Base Management

→ over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(DBM) by operating on the following four levels of data:

1) the user computer interface (Reserved Word);

2) the attribute and file or relationship (F/R) names (Data Name);

3) the modifiers of the attribute and F/R names (Data Descriptors); and

4) the occurrences of the attributes and F/Rs (Data Occurrence).

Hardware implementation designs are then considered for a subset of these functions and data levels. The data levels considered are the Data Name and Data Descriptor levels. Specifically, hardware designs are developed for the data and functions performed by a Data Dictionary and parts of a Data Directory. Given the proposed hardware implementation the final step in this research is to evaluate this hardware by comparing its processing times with the processing times for a conventional sequential computer.

The major conclusions drawn from this research are that the mathematical base not only provides a language for modeling DBM from the user's level down to the bit level but also provides a convenient method for implementing some of the DBM functions in hardware. The hardware evaluation can be generally interpreted to mean that the proposed hardware implementation is faster than the conventional method by a factor of two to twenty.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY OF TERMS

AAP	Associative Array Processor
A_o	a Normal DP set defined on $\overline{A}$
AL	the set of all attributes in a data base whose occurrences are represented as alphanumeric
a ℓ	alphanumeric representation for an attribute's values
AM	Associative Memory
AN	the set of AlphaNumeric characters
B	the set of all attributes in a data base whose occurrences are represented as boolean
b	boolean representation for an attribute's values
ß	represents a blank space
c.f.	Characteristic function
CMU	Control and Memory Unit
CPU	Central Processing Unit
DBM	Data Base Management
DBMS	Data Base Management System
DDP	Dictionary/Directory Processor
DPCS	Data Processing Characteristic Set
DP set	Data Processing Set

e.c.f.	extended characteristic function
F/R	File or Relationship
FIR	first inter-statement Normal DP set
F.P.	the set of all attributes in a data base whose occurrences are represented as floating point
f.p.	floating point representation for an attribute's values
I	the set of all attributes in a data base whose occurrences are represented as integers
I_j	Interrogation Line j
i	integer representation for an attribute's values
IA	Intra-statement Normal DP set
IR	Inter-statement Normal DP set
Job 1	given the F/R name, the DBMS is required to provide all its occurrences
Job 2	given an F/R name and a number of its attribute names, the DBMS is required to provide a subset of the occurrences of the F/R
Job 3	given an attribute's occurrence(s) of Type II synonyms, the DBMS is required to modify the attribute's and its synonym's occurrences in all their associated F/Rs
Job 4	given an attribute's occurrence(s) of Type I synonyms, the DBMS is required to modify the

	attribute's stored synonym occurrence(s) in its associated F/R
MIX	fictitious sequential computer
MIXAL	MIX's Assembly Language
N	the set of all positive integers, {1, 2, ...}
NTS	Normalized Time per MIXAL Statement
n_1	the number of F/Rs related to a given F/R and supported by the DBMS
n_2	the number of attributes in a given F/R
n_3	the number of associated F/Rs of a given attribute
n_4	the number of associated F/Rs of a given number (n_5) of attributes
n_5	the number of attributes in Job 2
n_6	the number of synonyms of a given attribute
$\dot{O}_i$	is a Normal DP set modeling the name of an F/R
P	the number of computer words per F/R name
P_1	the number of computer words per attribute name
P(X)	the power set of Set X
$P_\ell^i(X)$	Pull, an operator which subtracts the integer i from each element's superscript in the DP set, X
$P_s^i(X)$	Push, an operator which adds the integer i to each element's superscript in the DP set, X
R_j or r_j	Reset line j

R.T.	response time
$\overline{R.T.}$	average response time
RAM	Random Access Memory
S_j	set line j
SI_1 and SI_2	interrogation lines for the cells in ARRAY I and ARRAY IV
TA	Total number of attributes in a data base
TUMA	Transfer User's Memory Area
Type I Synonyms	the occurrences of the synonyms are stored in only one description and additional functions are developed to convert the different occurrences from one description to another
Type II Synonyms	the occurrences of the synonyms are stored in their different descriptions
U	a unit of time or relative measure
UMA	user's memory area
w	the number of computer words to be searched ($1 < w \leq 8$)
$\bigcup$	an OR gate
$\bigcap$	an AND gate
$\bar{X}$	an ordering or permutation with replacement of the elements contained in the set X
X^{-1}	the inverse function of the function X
$\lceil x \rceil$	the least integer $\geq x$
$\lfloor x \rfloor$	the greatest integer $\leq x$

	Reset/Set flip flop
$\sum_{i=j}^{t} X_i$	the summation of X_i's from X_j to X_t
$\binom{X}{Y}$	the combination of taking X things Y at a time

Chapter I

INTRODUCTION

In the years between 1950 and 1964, which Benjamin [3] and others refer to as the second generation of computers, programs were run in a "job shop" fashion. Each organizational function coded their own programs and utilized, maintained, and safeguarded their own data bases. With the advent of the third generation of computers, larger main memories, sophisticated operating systems, and procedural languages were more readily available. Organizations became aware of the benefits that could be incurred by combining their data for different functions (e.g. payroll, employee benefits, accounting, etc.). This new concept, that of different functions sharing the same data base, gave birth to the field of Data Base Management (DBM).

According to Fry [17], ". . . The original impetus for generalized processing came from the military because of the large volume of data that had to be processed quickly." He states that there were other motivations for the development of the Data Base Management System (DBMS) technology. These motivations are:

1) decrease the programming effort and lead time from data base application design to operational capability;
2) allow a non-programmer to interface with the data base;
3) decrease the effort required to respond to changing requirements; and

4) provide central control of the data bases.

Data base technology has now reached the point where an organization may buy, from software houses or computer manufacturers, off-the-shelf systems. According to Datapro Research Corporation [11]: "Today, data base management systems and data communications monitors are running neck and neck as the subjects that generate more inquiries to Datapro's Telephone Consulting Service than any other software topic." In conjunction with industrial sector accomplishments in the DBM field, the research and development sector has been working diligently at trying to solve some of the software and hardware problems. The software interested people have been looking at such topics as hashing algorithms, logical data models, optimum reorganization of physical structures, and search mechanisms.

The research addressed here is primarily oriented toward the hardware aspects of the DBM field, although it is somewhat concerned with the software area. The general problem is to determine the "functions" performed in DBM through modeling and then analyze how some of them might be implemented in hardware. In Chapter II, a selective set of significant papers is discussed. These papers are divided into two groups. The first group consists of papers pertaining to the modeling of data for DBM purposes. The second group consists of papers pertaining to those efforts concerned with hardware in the DBM field. Chapter III contains a description of the specific problem this research addresses and a methodology for its solution.

Presented in the fourth chapter is an attempt to derive a mathematical base for DBM which is applicable from the user's level down to the bit level of a digital computer. The fifth chapter contains illustrations of modeling DBM by using set theory and the developed mathematical base. Through this modeling, the "functions" of DBM can be discerned. The sixth chapter contains a multilevel description of a proposed hardware implementation of some of these "functions." Presented in the seventh chapter is a mathematical method that is utilized for evaluating the proposed hardware versus a sequential computer in the performance of four generic jobs of DBM. Finally, Chapter VIII's contents is concerned with the conclusions reached from this research and those areas where future research should be undertaken.

Chapter II

REVIEW OF THE LITERATURE

INTRODUCTION

Hardware in the field of Data Base Management (DBM) requires further investigation. This can be elaborated upon by the following quote by Berra [4]:

> Vast computer resources are required for the managing of large data bases. With hardware costs coming down, and software and personnel costs going up, it is important that one investigate the application of associative devices to the field of data base management to ascertain what gains might be made.

Su and Lipovski [38] sum up the situation in the following manner:

> The hardware limitations of conventional Von Neumann computers tend to straightjacket our approach to non-numerical processing in large data bases. Recent progress made in memory technology and integrated circuits has opened a door to new computer and memory device design and offers an opportunity to the non-numerical data processing profession to examine more closely what the problems are in data processing which are inherent in the existing hardware, why they are there and how they can be resolved or avoided if we are given the chance to design a new machine. In recent years, the cost of hardware is continuously decreasing whereas the cost of software does not enjoy the same benefit from technological progress. It seems to be perfectly reasonable to implement those frequently used software functions in hardware to increase the machine's operating efficiency and, at the same time, to eliminate the problems introduced by the mismatch of hardware with applications.

To accomplish what Su and Lipovski are suggesting, i.e. ". . . to implement those frequently used software functions in hardware. . .," these functions must first be rigorously defined.

Su and Lipovski do not spell out what these "frequently used software functions" are for non-numerical processing. The literature in the DBM field also lacks specifics about them. Since they are not generally known, in the required detail, a method for determining them must be settled upon before they can be implemented in hardware.

A methodology for determining these functions can be described as a modeling approach. The logic is to perceive or derive a mathematical language which can be used as a base to model DBM. Since, DBM is concerned with data as seen by the user communicating with a DBMS down to the bit representation of data stored within the computer and on its storage devices, it would be desirable if the mathematical language developed was useful in modeling the same span of data levels. It is believed that, through the process of modeling DBM, its functions or levels can be discerned in the required detail.

This modeling will also assist in determining future hardware. The functions discerned by this modeling can hopefully be used to classify those hardware that have been designed and/or constructed. The by-product of this classification is the knowledge of which functions have not been implemented in hardware.

The first portion of this chapter contains a literature review of a selective set of significant papers describing models developed for DBM. These models are reviewed to determine if one or more of them can provide the mathematical language or model described above. The second portion of this chapter provides a

literature review of some of the more significant papers concerning the hardware in the DBM field. These presented hardware descriptions will provide a baseline for future developments.

MODELS IN DATA BASE MANAGEMENT

Most of the modeling work in the DBM field has been concerned with either the lowest level of data or general concepts and specifications at the highest level. There seems to be a dearth of information on modeling that encompasses all levels. In this section a review of selected papers concerning levels of data and DBMSs is provided.

HIGH LEVEL MODELING

Representative reports at the highest level are the Data Base Task Group reports [8,9] and the Joint GUIDE-SHARE report [23]. As an example of this level, the Data Base Task Group report [8] provides a modeling of DBMSs. The modeling defines the tasks of a Data Administrator, the different levels of users, how to achieve privacy and integrity, and of course, data definition and manipulation languages. The relationships amongst data are discussed by describing the logical data structure of the sequential, tree, network, and cycle data structures. Different levels of data such as data-items, arithmetic data, string data, database-keys, vectors, repeating groups, records, etc. are also defined. These reports, which pertain to the highest level of DBMS, do not go into any discussion of how a DBMS should or could be implemented.

They also do not provide the detail necessary to model DBM as discussed above. Hence, they were not used as the base for the mathematical language desired.

DATA RELATIONSHIP MODELING

Two examples of another level of modeling in DBM can be found in Child's [7] and Codd's [10] papers which address the relationships of data. Codd addresses the data relationship area by defining a relationship as a subset of a cartesian product of n sets. Each set can be looked at as being attributes in a data base. Codd defines the concepts of normalized and unnormalized sets, the operations of projection and join, composition, and strong and weak redundancy relationships. These concepts and their operations are defined and discussed at a very high level. They are not interfaced with a higher or lower level of implementation, e.g. how the user would interface with these concepts and how they could physically be implemented on a computer.

Child's paper also addresses the data relationship area. His approach is based on set theory. Child states: ". . . any relation can be expressed in set theory as a set of ordered pairs and since set theory provides a wealth of operations for dealing with relations, a set-theoretic data structure appears worth investigating." He recognizes that order in relationships, say <a,b> can be represented as i.e. <a,b> = {a, {a,b}}; but that they become difficult to handle as the relationships become greater than binary. To be able to maintain sets and set theory to model

data structures and to keep the concept of order necessary for describing relationships, Child introduces what he defines as a complex:

"Definition 1. Any two sets A and B form a complex (A;B) iff $(\exists X)\ (\exists Y)\ (X \epsilon \{A,B\})\ (Y \epsilon \{A,B\})\ [(\forall x \epsilon X)\ (\exists i \epsilon N)\ (\{\{x\},i\} \epsilon Y)$ and $(\forall y \epsilon Y)\ (\exists j \epsilon N)\ (\exists x \epsilon X)\ (\{\{x\},j\} = y)]$," where N is the set of natural numbers.

The following is an alternate manner of defining the concept: Any two sets A and B form a complex (A;B) if and only if there exists a set X and a set Y such that X is an element in the set {A,B} and Y is an element in the set {A,B}. In conjunction with the above, for all elements x contained in X there exists an element i contained in N (the set of natural numbers), such that the set containing the set {x} and the element i, i.e. {{x},i}, is contained in the set $Y(\{\{x\},i\} \epsilon Y)$. Similarly, for all elements y contained in Y there exists an element j contained in N and there exists an element x contained in the set X such that the set containing the set {x} and the element j, {{x},j} is equal to y.

An example of a complex can be constructed. Let X = {T,A,C} and Y = {{{A},2},{{C},1},{{T},3}}}. Then the complex (X;Y) is equal to ({C,T,A}; {{{A},2},{{C},1},{{T},3}}). The rest of this paper deals with the proving of theorems and the defining of additional concepts involving complexes.

Like Codd's paper, the level of Child's work does not venture from that of set theory. No attempt is made to show how the

modeling language applies in describing for instance, the implementation process of building a DBMS. Child's paper concludes by describing two example data bases and the time required to perform some simple queries to the data.

Of these two papers, Child's concept of complexes has the highest potential of providing the mathematical language necessary for the purposes previously specified and desired. However, it was not used because additional concepts involving complexes hindered the logic needed for the model desired. Codd's model is described at only one level of DBM and therefore was too limited for the mathematical model required and hence, was also not used.

DATA INDEPENDENT ACCESSING MODEL

Another level of modeling DBMSs is the Data Independent Accessing Model (DIAM) described by Senko, Altman, Astrahan and Fehder [36]. DIAM is composed of four models. They are the Entity Set Model, the String Model, the Encoding Model, and the Physical Device Level Model.

The Entity Set Model is at a very high level. It deals with the entry mechanism between the world of tangible information external to the data base system and the DIAM. Therefore, information can be stored in a DBMS by describing the information in terms of the Entity Set Model Name Organization. DIAM will catalogue the information as described by these terms. The basic building block of the model is a triplet of Entity Name Set Name/ Role Name/Entity Name, where Entity Name is drawn from the "Entity

Name" Set. An example of this triplet concept would be to have "Part" as an Entity Name Set Name, "Part Supplied" as a Role Name and "Gear" as an Entity Name. Therefore the triplet would be Part/Part Supplied/Gear. The authors believe that Entity Names and Entity Name Set Names are more useful building blocks of structured information than fields, records, and files.

The next lower level model is the String Model. The String Model is used to describe how to traverse within or through Entity Sets. The String Model is broken down into A-strings, E-strings, and L-strings. A-strings are defined within the same Entity Set Description by order. E-strings are defined between the same Set Description by order. L-strings are not restricted, but connect elements based on a match between Entity names for the same Entity that occurs in each of the elements (which may be an A-string, an E-string, or another L-string).

The Encoding Model provides a bit level representation for the strings in the String Model. The heart of the Encoding Model is the Basic Encoding Unit (BEU). The BEU provides one basic format for encoding all strings and triplets.

The lowest level modeled is the Physical Device Level Model. This level models the placing of Contiguous Data Groups (CDG), which are similar to data records, onto a physical device (e.g. disk or drum). A CDG is made up of a set of BEUs. As stated by the authors the DIAM, like many other models of data base systems, ". . . does not cover all aspects of system description, but it does appear to describe and provide defined, detailed interfaces to a broad range of components of data base systems."

DIAM provides an excellent set of models for describing data, their relationships within a DBMS and some DBM components. However, it does not appear to provide any mathematical model with a language that can easily be extended to all aspects of DBM.

ATTRIBUTE BASED MODEL

The last modeling technique is probably the one most known to the data base community. It is described by Hsiao and Horary [22]. This is the Attribute Based Model where there exists ". . . a set A of "attributes" and a set V of "values." A "record" R is a subset of the Cartesian Product A x V in which each attribute has one and only one value." Thus, R is a set of ordered pairs of the form: (an attribute, its value). Stated later in the paper is that a file is a set of records. Given these definitions and other basic concepts (such as indices, key words, (A x V) ordered pairs, and key word lists), the authors modeled inverted files, multilist files, and index sequential files. They then proceed to define two functions (a directory search function and a file search function) and develop within their model a Simple Retrieval Algorithm (serial processing of lists) and a General Retrieval Algorithm (parallel processing of lists). Their modeling, like that of Codd and Childs, stays at one level and no attempt is made to link it with any upper or lower levels. Thus, this modeling was not further investigated to determine whether or not it fulfilled the requirements for the desired mathematical model.

SPECIAL HARDWARE AND DATA BASE MANAGEMENT

Some researchers have recognized the importance of investigating special machines for non-numerical processing. These alternate machines have included Associative Memories (AM), Associative Array Processors (AAP) and special hardware for intersecting and combining sets or lists of data. The majority of this hardware deals with the data in the data base or, more specifically, the occurrences of files or relationships. The descriptions of these machines provide a baseline for future hardware designs. An overview of other developed hardware can be found in Thurber's paper [39].

ASSOCIATIVE MEMORIES AND PROCESSORS

Some of the work of applying associative devices to DBM was directly concerned with associative memories and processors. These research efforts were conducted by DeFiore and Berra [13,14], Moulder [31], and Linde, Gates and Peng [28]. DeFiore and Berra [14] developed mathematical equations for analyzing an inverted list implementation of a data base against an associative memory (AM) implementation. A basic assumption that DeFiore and Berra made was that the data base was totally contained within the AM. This assumption was made because as the data base becomes larger than one AM load, the system may become I/O bound and therefore will not utilize the AM hardware to its uppermost capability. Moulder [31] recognized this problem and proposed a system consisting of a SIGMA 5 sequential computer, a STARAN AAP and a parallel

head per-track disk (PHD) with a capacity of 12,288 bytes/surface. The system was designed such that two revolutions (39 msec/ revolution) of the disk allowed a user to process a simple job pertaining to the total data base. However, as the data base grows larger than one surface of the PHD, the number of revolutions increases by two revolutions per surface.

Linde, Gates, and Peng [28] proposed a system with more power than the above mentioned work. They compared a hypothetical machine against the IBM 370/145 for the data management jobs of data retrieval, update, and search. Their machine differed from the others in that it was byte-serial, word parallel rather than bit-serial, word parallel. The other major difference in this machine is that unlike the PHD, it was connected to a large (500,000 bytes) hypothetical random access data memory with a transfer rate of 1.6 billion bytes/sec.

DISTRIBUTED LOGIC

The above research is representative of the kind of work being done in "conventional" AMs/APs in DBM. The common thread of this work is that data are processed within the AM/AP. Another approach being looked at is what might be called distributed logic systems. The main ingredient of this approach is the placement of logic on the secondary storage device. This allows for amounts of data to be searched in place rather than moving the data into main memory for searching.

Work by Parker [34] and Minsky [30] discuss logic on rotating devices. Both of their architectures are designed to search on

only part of the data base. Therefore, they are sometimes called "partially associative memories." These architectures differ in at least two major ways:

1) Parker's structure can have keys, holes, and data on the same track while Minsky's structure has keys on a separate disk from the data, and
2) Parker's structure allows for variable length keys, holes, and data while Minsky's structure has fixed size cells for keys and data.

In addition to the above there is ongoing work in "fully associative memories" that utilize rotating devices. Some of this work is being performed by Parhami [35], Healy, et al. [20] (see also [6, 15, 19]), Ozarahan, et al [32, 33] and Lin, et al. [27]. Their work, except for Parhami's is partly described in the names given to their architectures. Healy's architecture is named CASSM for a Content Addressed Segment Sequential Memory, Ozkarahan's architecture is named RAP for a Relational Associative Processor and Lin's architecture is named RARES for a Rotating Associative Relational Stores. In addition to rotating devices there is a distributed logic system design named ECAM [1] for an Extended Content Addressed Memory.

In Parhami's structure the data are loaded and read in a word-serial bit-parallel fashion. The bits of a character are loaded on parallel tracks. Since there is a head per track, it is implied that data are read bit-parallel and word-serial.

In the CASSM and RAP structures the data are stored serially on a track. The tracks are read simultaneously. Therefore, in

reference to Parhami's structure, this can be referred to as word parallel. In the CASSM structure microcode instructions are loaded with the data on secondary storage. The RAP structure stores and maintains its data directory information on its secondary storage. This secondary storage for the RAP system is currently being changed from a head per track device to charge coupled devices.

The RARES system stores its data bit-serial and byte-parallel. The domains in a data relation are stored serially and in parallel on its rotating device. For instance, three parallel tracks may contain the domain that is three bytes long. The next domain in a relation may be on the next three or four parallel tracks followed serially by another domain in the relation. This configuration is called an "orthogonal storage lay out." It reportedly [27] "allows a high output rate of selected tuples to be attained even when a sort order must be preserved."

Finally, there exists a distributed logic system design called ECAM. It is being designed as a special purpose machine (with a storage capacity on the order of 10^9 bits) to be attached to one or more host computers. It is structured for relational data bases and will have a reportoire of associative search and arithmetic operations. The content addressable memory (currently considering charge coupled devices) will contain up to 250,000 associative words of 4,096 bits in length.

SEGMENTED ASSOCIATIVE MEMORIES

Another approach is that of segmented associative memories by Love [29]. Instead of placing logic on a disk or drum, Love

divides the logic into 10 AMs which share a data base stored on a shift register bulk memory. There are two processors which manage data and the AMs. Control processors manage the data and instruction processors manage the AMs. As stated by Love, this network structure ". . . permits each associative memory to be assigned to a data transfer channel for bulk memory, and also permits the associative memories to be connected together in parallel in various combinations." He goes on to state that, "This capability makes it possible for several of the associative memories to operate as a single large associative memory when the amount of data requires it, or to operate individually in simultaneous independent operation."

SUPPORT HARDWARE

In all of the above discussed literature, one common thread has been that the hardware is essentially dealing with the data in the data base i.e. the occurrences of files or relationships. Other designed hardware exists. These hardware are designed more for supporting a DBMS in performing its functions rather than manipulating its data. Three examples in this area are hardware structures posed by Baum and Hsiao [2], Hollaar [21], and Singhania and Berra [37,5]. Baum and Hsiao have developed a hardware architecture for an attribute oriented DBM machine whose primary concern is to provide data security. The hypothetical machine is made up of a directory memory, an intersector, a mass memory, and a command pre-processor. The directory memory contains

directory information. The mass memory contains the data base itself. The intersector is the directory memory processor. One of its main functions is to intersect sets of data from the directory memory. The command pre-processor is that element which directs all the other components.

Hollaar's research was performed in the context of an information retrieval system [21]. Many of these systems employ inverted files. His work yielded a hardware structure for combining ordered lists which can be used on ordered inverted files.

Singhania's and Berra's work is related to a hardware implementation of a data directory for a very large data base. The architecture consists of pipelining an N-level hierarchial directory based on multiple keys. Each level of the hierarchy is processed by an associative memory. The architecture is designed to provide concurrent processing, i.e. processing within an AM and within the pipeline.

CONCLUSION

It is not obvious that any one of the above hardware devices will solve all the problems that will occur in developing a DBMS for a large integrated data base. This feeling was also reflected by Berra [4], ". . . it appears at the present time that associative memories and processors

have a potential for reducing some of the pressing problems in the field but they are by no means the final answer; only a step on the way to more sophisticated devices." He goes on and states that,

> . . . there are thousands of data base problems in existence today that would support the development of computers strictly for the solution of these problems. Imagine the vast amounts of data the various government agencies must manage, let alone all of the industrial organizations and businesses that are aspiring to integrated corporate data bases. Imagine also the vast amount of computer resources that are wasted in processing largely non-sequential data on sequential computers.

Similar words by Su and Lipovski were cited earlier in this chapter. However, they do not say how one determines which software functions, if implemented in hardware, will increase a machine's operating efficiency or how one determines how to find these elusive software functions. Berra [4] sums it up when discussing the future, ". . . It seems clear to this author that for the next few years the associative device will remain essentially a peripheral to a sequential computer. One reason is that we just don't know enough about the generic functions that must be performed in data base management and therefore can't really define what we need from the hardware." Therefore, the problem is to first define these generic functions and then implement them in hardware. Once this is completed a DBM computer will become a reality.

Chapter III

PROBLEM DEFINITION AND METHODOLOGY

INTRODUCTION

In the previous chapter a review was presented of a selective set of significant papers describing the modeling of data for DBM and those architectures being built and investigated for the DBM field. The papers discussing the modeling of the data were presented to determine if any of their approaches could be used in defining the generic functions of DBM. The contents of the remaining papers provided a baseline for the functions of DBM that have been addressed by their implementation in hardware. It was also pointed out that if a DBM computer is to be a reality, then the generic functions of DBM must first be defined.

Provided in this chapter is a definition of the specific problem this research addresses and a methodology for its solution. The methodology is presented as an overview describing the four phases of the solution procedure. Following this overview, each phase is discussed at a more detailed level.

PROBLEM DEFINITION

Succinctly stated, the problem addressed in this research is to first develop a mathematical modeling concept that can be utilized to model DBMSs from the user level down to the bit level

and secondly to utilize this mathematical base to define in detail some of the functions that must be performed in DBM. Utilizing the definition of these functions the next step is to consider the implementation of them in hardware. Given a proposed hardware implementation, the final step in this research is to evaluate the hardware and compare it with more conventional approaches.

METHODOLOGY OF SOLUTION

The method used to approach the above problem can be divided into four phases. The development of a mathematical language for possible use as a base for modeling DBM is dealt with in the first phase. It is required that the language be applicable from the user's view of the data down to the bit representation of the data occurrences. Phase 2 encompasses the application of this mathematical language. This modeling procedure yields the levels of DBM and some of its functions. In the third phase the levels or functions that have been considered for implementation in a hardware design are compared with those discerned from the second phase. A hardware design is developed for a subset of those levels and functions. The design is evaluated in the fourth phase by comparing the hardware time with the time required for a conventional sequential computer to perform a number of functions of DBM.

PHASE 1: A MATHEMATICAL BASE FOR DATA BASE MANAGEMENT

In the previous chapter, the literature survey of the significant papers describing models developed for DBM indicated that

none of them provided the desired capability. However, Child's model, applying the concept of order to sets, came the closest and therefore was pursued in the expectation that a mathematical base for modeling DBM could be obtained.

The properties of this base are similar to the properties of set theory. For example, the properties of containment, union, and intersection are developed. In the process of developing these properties, new operators are required and hence are defined.

The relationships between sets and sets with order are developed to finalize the mathematical language. This entails the derivation of how sets with order are defined from sets and their inverse relationship. The next level of relationships derived is concerned with characteristic functions for sets and for sets with order. Characteristic functions are investigated because they provide a convenient method of implementing this mathematical language in hardware.

PHASE 2: DATA BASE MANAGEMENT AND SETS

The mathematical language is used to model DBM levels and functions. DBM levels are divided into four parts and can be considered as being static. These four parts are:

1) the user computer interface;
2) the attribute and file or relationship (F/R) names;
3) the modifiers of the attribute and F/R names; and
4) the occurrences of the attributes and F/Rs.

DBM functions are not divided into any specific parts. They are

dynamic and operate on the above levels to perform the jobs requested by a user of a DBMS.

PHASE 3: HARDWARE

The design procedure is pursued in four non-disjoint steps. The first step interfaces the hardware with a sequential computer. Next, a design of the hardware at a more detailed level considering its control, data storage, and data transfer is performed. The hardware at the logic gate and flip flop levels is designed in the third step. The final step insures that the design in each of the above three steps will perform the desired functions.

PHASE 4: HARDWARE EVALUATION

The method used to accomplish the hardware design evaluation can be divided into five steps. The development of the timing equations for sub-functions is accomplished in the first step. These sub-functions can be thought of as "microfunctions" that can be put together to form a DBM function. Defined next are a number of job types that can be submitted by a user to a DBMS. Together, these job types entail all functions that the hardware would have to perform in a "real" environment. Timing equations are then developed for each of these job types by summing the timing equations for the proper microfunctions. In step three the data structure and software techniques necessary to perform the same functions implemented on a sequential computer are designed. Development of the timing equations for each of the above job types, on the

sequential computer is performed next. The fifth step compares the timings for each of the job types performed on the hardware and the sequential computer.

SUMMARY

This chapter contained the definition of the problem this research addresses, and the methodology utilized in solving this problem. The methodology was described as a four-phase overview with each phase being discussed at a more detailed level. The next four chapters contain the details of each of the four phases respectively.

Chapter IV

A MATHEMATICAL BASE FOR DATA BASE MANAGEMENT

INTRODUCTION

The objective of this chapter is to provide a mathematical base that can be used to model data base management and its implementation in hardware. The chapter centers around ten properties of sets. These properties are defined early in the chapter. This is followed by the development of a concept called Data Processing Sets which are sets with order. Then ten similar properties are defined for Data Processing Sets. The next portion of the chapter involves the relationships between sets and Data Processing Sets. This provides a background on the similarities and differences between the two concepts.

Data Processing Sets provide a mathematical base for the modeling of DBM at the user's level. To extend this mathematical base such that it can be utilized for modeling DBM at a digital level and/or to design computer hardware requires the introduction of Data Processing Characteristic Sets. The final portion of this chapter defines the above ten properties, with examples, for Data Processing Characteristic Sets.

A REVIEW OF SETS AND THEIR PROPERTIES

A set may be thought of as a collection of elements. They will usually be denoted by one or more upper case letters.

However, a set's elements will usually be indicated by lower case letters and may be enclosed by { }. An example of a set and its elements is:

$$X = \{x_1, x_2, \ldots, x_n\},$$

where X is a set and $x_1, x_2, \ldots, x_n$ are the elements contained in X.

Some of the various properties of sets that are used throughout this research are presented below with the aid of the defined symbols in Table IV-1.

I Subset (or Containment)

$$X \subseteq Y \longleftrightarrow x \in X \rightarrow x \in Y$$

II Proper Subset

$$X \subset Y \longleftrightarrow (X \subseteq Y) \wedge (\exists(x \in Y) \ni x \notin X)$$

III Set Equality

$$X = Y \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$$

IV Set Intersection

$$[X \cap Y = Z] \longleftrightarrow [(x \in Z) \longleftrightarrow (x \in X) \wedge (x \in Y)]$$

V Set Union

$$[X \cup Y = Z] \longleftrightarrow [(x \in Z) \longleftrightarrow (x \in X) \vee (x \in Y)]$$

VI Set Difference

$$[X - Y = Z] \longleftrightarrow [(x \in Z) \longleftrightarrow (x \in X) \wedge (x \notin Y)]$$

VII Power Set

A power set of a set is a set containing all the subsets

Table IV-1

Set Theory Notation

SYMBOL	MEANING
$X \subseteq Y$	X is a subset of Y
$X \subset Y$	X is a proper subset of Y
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y (inclusive)
$X \rightarrow Y$	X Implies Y
$X \longleftrightarrow Y$	(X Implies Y) AND (Y Implies X)
$x \in X$	x is contained in X
$x^j \in X$	x^j is contained in the DP set X
$X \cup Y$	X Union Y
$X \cap Y$	X Intersect Y
$X - Y$	X And Not Y (Difference)
$\forall$	For All
iff	If and only if
$\exists$	There exists
$\ni$	Such that
Φ	Null Set
$x \notin X$	x is not contained in X
$x^j \notin X$	x^j is not contained in the DP set X
$a^i_j(\epsilon Z) = 0$	a^i_j, an element of the set Z is equal to zero.

of the original set. That is, for every set X, the set of all subsets of X, called P(X), is the power set of X. Symbolically,

$$P(X) = \{Y: Y \subseteq X\}.$$

For example, if $X = \{x_1, x_2\}$,
then $P(X) = \{\Phi, \{x_1\}, \{x_2\}, \{x_1, x_2\}\}$.

VIII Power of a Set

The power of a set is the number of unique elements contained in the set. For example, the power of sets $X = \{x_1, x_2\}$ and $Y = \{x_1, x_2, x_2\}$ is two.

IX Power of a Power Set

If the power of a set X is T then the power of a power set X (P(X)) is 2^T. For example, if $X = \{x_1, x_2\}$, then the power of X is two and the power of the power set of X (P(X) = $\{\Phi, \{x_1\}, \{x_2\}, \{x_1, x_2\}\}$) is $2^2 = 4$. This can be shown by noting that the power set is the set of all the subsets of the original set plus, of course, the null set (which is a subset of every set). Therefore, if the power of the original set is T, then the number of elements in its power set is the sum of the combinations of T elements taken one at a time, two at a time, ..., T at a time, plus the null set, i.e.

$$\sum_{i=1}^{T} \binom{T}{i} + 1 = 2^T.$$

X Characteristic Function of a Set

The domain of a characteristic function (c.f.) for a set X is the power set of X. Its range is the set {0,1}. The value of c.f. of some element of X related to some element of P(X) is dependent on whether or not that element of X is contained in the element of P(X). This can be shown as:

$$\text{c.f.}: X \rightarrow \{0,1\} \text{ where } \text{c.f.}_{X_i}(x_j) = \begin{cases} 1 & \text{if } x_j \in X_i \\ 0 & \text{if } x_j \notin X_i \end{cases},$$

$X_i \subseteq X$ and $x_j \in X$.

Consider an example: $X = \{x_o, x_1, x_2\}$ and $X_1 \subseteq X$ where $X_1 = \{x_o, x_1\}$. Then the c.f., with respect to X_1, is equal to $\text{c.f.}_{X_1}(x_o) = 1$, $\text{c.f.}_{X_1}(x_1) = 1$, and $\text{c.f.}_{X_1}(x_2) = 0$. Note from this example that the inverse of this characteristic function (c.f.^{-1}) is not a function since it is not one-to-one.

DATA PROCESSING SETS

If a mathematical base is going to model DBM it should at least be capable of modeling the Input/Output to a DBMS. This was a major driving factor in the development of Data Processing (DP) Sets. Consider trying to model the marks made across this page as a set. For each advancement of the typewriter carriage the typewriter can place only one character on the page, but each character may appear in more than one position on a line.

If two words or two lines of print are identical, then not only must the characters be the same but their respective positions must also be the same. This mode of thinking has led to the basis of the proposed mathematical base.

Imagine trying to model the lines of type by using set theory. If the following two phrases

1) "she is smarter than him"

and

2) "he is smarter than her"

are modeled as "sets," (ignoring blanks) then they become equivalent by the Axiom of Extent [40]. That is,

{s,h,e,i,s,s,m,a,r,t,e,r,t,h,a,n,h,i,m} = {s,h,e,i,m,a,r,t,n},

{h,e,i,s,s,m,a,r,t,e,r,t,h,a,n,h,e,r} = {h,e,i,s,m,a,r,t,n} and

{h,e,i,s,m,a,r,t,n} = {s,h,e,i,m,a,r,t,n}.

However these "sets" or relations in the real world are not equal because:

1) there are a different number of occurrences of elements (e.g. s occurs three times in the first phrase and twice in the second phrase); and

2) the order of the elements in the two phrases is not the same.

"Sets" which have these properties, plus the additional property that two or more elements cannot occupy the same position in a set shall be called DP sets. An element of a DP set shall have an integer superscript greater than zero. The value of the superscript designates the order (or position) of the element in

the DP set. An alternate notation will sometimes be used where the element's superscript value will be implied, by the element's position in the DP set, from one to the number of elements in the DP set. For example, let X and Y be DP sets where

$$X = \{ x_2^1, x_1^3, x_3^2, \ldots, x_n^k \},$$

and

$$Y = \{a,b\}.$$

The DP set X is interpreted as having element x_2 being the first element in X, x_1 being the third element in X, x_n being the k-th element in X, etc. The DP set Y is interpreted as having a as its first element and b as its second element.

A DP set is a set of elements where each element has two parameters. One parameter is the "value" of the element (e.g. x_1, a, y_2, z, etc.) and the other parameter is the position the element occupies in the DP set. In a DP set two or more elements may have the same value. Consider the following unequal DP sets,

$$\{d^1, a^2\} \neq \{a^1, d^2, d^3\}.$$

The d^2 and d^3 elements of $\{a^1, d^2, d^3\}$ have the same value d. However, two or more elements of a DP set cannot occupy the same position. For example, $X = \{a^1, d^2, d^2\}$ and $Y = \{d^1, a^1\}$ are <u>not</u> DP sets. Therefore, an element x^j is contained in a DP set if and only if the j-th position of the DP set has an element with value x and j is an integer greater than zero. Stated in another way, let $N = \{1, 2, \ldots\}$, be the set of positive integers, and if x^j is an element of a DP set X, then this implies if there is

another element in X, say y, which has a superscript j then $x = y$.

Given the above, a formal definition of a DP set can be stated. X is a DP set provided that:

1) $\exists\, x^j \in X$;

2) $j \in N$; where $N = \{1, 2, \ldots\}$; and

3) if $y^j \in X$ then $x = y$.

PROPERTIES AND OPERATORS OF DATA PROCESSING SETS

To utilize DP sets in the modeling of DBM, some basic properties (similar to those contained in set theory) and operators must be defined. Referring again to the symbols in Table IV-1, consider the following properties and operators for DP sets.

XI DP Set Subset (or Containment)

$$X \subseteq Y \longleftrightarrow (x^i \in X \rightarrow x^i \in Y)$$

XII DP Set Proper Subset

$$X \subset Y \longleftrightarrow (X \subseteq Y) \wedge (\exists (x^i \in Y) \ni x^i \notin X)$$

XIII DP Set Equality

$$X = Y \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$$

XIV DP Set Intersection

$$[X \cap Y = Z] \longleftrightarrow [(x^i \in Z) \longleftrightarrow (x^i \in X) \wedge (x^i \in Y)]$$

In order to define the union and difference properties of DP sets, two new operators must first be defined. These operators

are called Pull (P_ℓ^i) and Push (P_s^i). $P_\ell^i(X)$ is defined as an operator which subtracts the integer i from each element's superscript in X. $P_s^i(X)$ is defined as an operator which adds the integer i to each element's superscript in X. However, note that if the value of an element's superscript is equal to or less than zero, then the element is <u>not</u> contained in X; since the superscript must be contained in N. Consider the following example.

$$Y = \{a^s, b^r, a^t, \ldots, u^q\}$$

then

$$P_s^r(Y) = \{a^{s+r}, b^{r+r}, a^{t+r}, \ldots, u^{q+r}\}$$

and

$$P_\ell^r(Y) = \{a^{s-r}, a^{t-r}, \ldots, u^{q-r}\}.$$

The union and difference properties can now be defined.

XV DP Set Union

$[X \cup Y = Z] \longleftrightarrow [(x^i \in Z) \longleftrightarrow (x^i \in X) \vee (x^i \in Y)]$

An example of using the push operator and the union property would be in expressing the DP set {a, d, d} as the union of $\{a^1\}$ and $\{d^1, d^2\}$ i.e.

$$\{a^1\} \cup P_s^1\{d^1, d^2\} = \{a^1, d^2, d^3\}.$$

XVI DP Set Difference

$[X - Y = Z] \longleftrightarrow [(x^i \in Z] \longleftrightarrow (x^i \in X) \wedge (x^i \notin Y)]$

An example of the pull operator and the difference property would be in expressing the DP set $\{d^2\}$ as the difference of $\{d^5, d^8\}$ and $\{d^5, d^7, d^9, d^{10}\}$, i.e.

$$P_\ell^3\{d^5, d^8\} - \{d^5, d^7, d^9, d^{10}\} = \{d^2\}.$$

XVII DP Set Power Set

A power set of a DP set is identical to property VII for sets, i.e. if X and Y are DP sets then

$$P(X) = \{Y: Y \subseteq X\}$$

XVIII Power of a DP Set

The power of a DP set is the number of unique elements contained in the set. For example, the power of DP sets $X = \{x_1^1, x_2^2\}$ and $Y = \{x_1^1, x_2^2, x_2^3\}$ is 2 and 3 respectively. (See the example given for property VIII.)

XIX Power of a DP Set Power Set

The power of a DP set power set is identical to property IX for sets, i.e. if the power of a DP set is T then the power of its DP set power set is 2^T.

XX Characteristic Function of a DP Set

The domain of a characteristic function (c.f.) for a DP set X is the power set of X. Its range is the set (0,1). The value of c.f. of some element of X related to some element of P(X) is dependent on whether or not that element of X is contained in the element of P(X).

$$\text{c.f.: } X \to (0,1) \text{ where } \text{c.f.}_{X_i}(x_j^i) = \begin{cases} 1 & \text{if } x_j^i \in X_i \\ 0 & \text{if } x_j^i \notin X_i, \end{cases}$$

$X_i \subseteq X$ and $x_j^i \in X$.

DATA PROCESSING SETS AND SETS

The contents of the next chapter are concerned with the modeling of DBM. This modeling utilizes the above properties and operators in a more realistic manner than the above examples. Along with DP sets, the modeling also employs sets and their properties. To provide a better understanding of this modeling, the relationships between sets and DP sets are discussed below.

The concept of DP sets is actually an extension of set theory which adds the parameter of position to the elements in a set. By removing this extension there exists a one-to-one relationship between a DP set and a set. That is, given any DP set there exists a function (f) that maps this DP set to one and only one set. The domain of f is a DP set and the range of f is a set. (Consider f as a function that removes the position parameter imposed on the elements of a DP set.)

Let A_1 be a DP set and A a set where A_1 is the domain of f and A is the range of f. Then f is defined in such a way that if $f(x^i) = y$ then $x^i \in A_1$, $y \in A$, and $x = y$.

For every DP set A_1, the function f maps its elements into one and only one set A. Let this set A be called the Basis set for the DP set A_1. Therefore, a Basis set can be defined as:

A is a Basis set of a DP set A_1 if and only if for every $(x^j \in A_1)\exists a(y \in A)\ni(x = y)$ and if $(y \in A)\exists$ one or more $(i \in N) \ni (x^i \in A_1)$ and $x = y$.

It should be noted that although there is only one Basis set for each DP set, more than one DP set may have the same Basis set.

To illustrate the above concepts, consider the following examples:

Example 1: Let $A_1 = \{a^1, d^3, d^2\}$

then $f(a^1) = a$,

$f(d^3) = d$,

and $f(d^2) = d$.

Example 2: Let $A_1 = \{a^1, d^2, d^3\}$,

then the Basis set of A_1 is $A = \{a, d\}$.

Example 3: Let $A_1 = \{a^1, d^2, d^3\}$ and $A_2 = \{d^1, a^2\}$

then the Basis set for A_1 is $\{a, d\}$ and the Basis set for A_2 is $\{a, d\}$. Note that $\{a, d, c\}$ is not a Basis set for either A_1 or A_2 since there does not exist a $j \in N$ such that $c^j \in A_1$ or $c^j \in A_2$.

DEFINING PROCEDURE FOR DATA PROCESSING SETS

The above concepts of power sets and the power of a set are needed to help describe a procedure for defining DP sets. This procedure can best be expressed by first recognizing the existence of a non-empty set A. Then define a subset of A (say X) as a Basis set where X is an element of the power set of A($X \in P(A)$). Given the elements in X, an ordering or permutation with replacement $\overline{X}$ can be defined. (Note that $\overline{X}$ is not defined as a set.) The last step is to define on $\overline{X}$ a DP set Y. An example to illustrate this procedure is now given. Consider the keys on a typewriter. Let the available symbols be defined as a set

AlphaNumeric (AN) = {A, B, ..., 0, 1, ..., ?, ...,]}. Let X be a Basis set where X ϵ P(AN) = {C, P, A, R, O}, $\overline{X}$ = CAPRARO and therefore the DP set Y = {C^1, P^3, O^7, R^4, R^6, A^2, A^5}.

It can be seen from the information provided thus far that given a non-null Basis set X, an infinite number of DP sets can be defined. This is true since there are no restrictions on the power of the Basis set, the power of the DP set and the limit of the values of the element's superscripts in the DP set. For the modeling purposes to be discussed in the next chapter, this infinite upper bound is generally unrealistic. This can be rectified by placing a finite limit k on the power of the Basis set, a finite limit T on the power of the DP sets, and restricting the DP sets to be what is called a Normal DP set. A Normal DP set is a DP set whose minimum superscript value is one and whose maximum superscript value is equal to the power of the DP set. For example (a^1, b^2) is a Normal DP set but (a^2, b^3) is not a Normal DP set.

Given the above restrictions, if a Basis set X has a finite power of k and the Normal DP sets defined with X as a Basis set have a power equal to or less than T (where T is finite), then the maximum number of Normal DP sets that can be defined is

$$\sum_{i=1}^{T} k^i.$$

For example, let X = {A} and T = 2. The maximum number of Normal DP sets that can be defined on the Basis set X is

$$\sum_{i=1}^{2} 1^i = 2.$$

These Normal DP sets are {A} and {A,A}. The expression k^i is obtained by viewing the situation as the placing of k things in i boxes with replacement, i.e. there are k ways of filling box 1, k ways of filling box 2, ..., k ways of filling box i. Therefore there are k^i ways of filling i boxes with k things, with replacement.

Extending this concept further, a super DP set of a Basis set X can be defined as:

A super DP set of a Basis set X is that set containing the null set and all the possible Normal DP sets that can be defined on X. For the examples above with k and T finite 1 and 2 respectively and the DP sets normal the super DP set of X is $\{\Phi, \{A^1\}, \{A^1, A^2\}\}$. The power of a super DP set can be determined by

$$\sum_{i=0}^{T} k^i.$$

For this example the power is

$$\sum_{i=0}^{2} 1^i = 3.$$

DATA PROCESSING CHARACTERISTIC SETS

One of the major thrusts of this research is to show that DP sets and sets can be used to model DBM from the user's level

of data to the bit representation of data. So far, the mathematical bases for modeling the user's level of data has been presented. To extend this mathematical base to the bit level requires an extension to the property called the characteristic function of a DP set (property XX).

An extended characteristic function (e.c.f.) can be defined for DP sets whose inverse is also a function. Following the notation above let 1) A be a Basis set whose power is y, 2) $\bar{A}$ be defined as one of the y! possible permutations of the elements of A, 3) A_o be defined as a Normal DP set defined on $\bar{A}$ and 4) a Normal DP set whose elements have values of zeros and/or ones be defined as a Data Processing Characteristic Set (DPCS). Now an e.c.f. can be defined as:

$$\text{e.c.f.: } A_o \rightarrow \text{DPCS where e.c.f.}_{A_i}(a_j^i) = \begin{cases} 1^i & \text{if } a_j \in A_i \\ 0^i & \text{if } a_j \notin A_i, \end{cases}$$

where $A_i \subseteq A$, (Power of {DPCS}) = (power of A) = y, and $a_j^i \in A_o$. The domain of this e.c.f. is the power set of A. Its range is a set of 2^y elements (See properties IX and XIX) each being a DPCS. For an example of the above definition of an e.c.f., let:

$A = \{a_1, a_2, a_o\}$ $\quad A_2 = \{a_o, a_1\}$ $\quad A_6 = \{a_1\}$

$\bar{A} = a_o, a_1, a_2$ $\quad A_3 = \{a_o, a_2\}$ $\quad A_7 = \{a_2\}$

$A_o = \{a_o^1, a_1^2, a_2^3\}$ $\quad A_4 = \{a_1, a_2\}$ $\quad A_8 = \{a_1, a_2, a_o\}$

$A_1 = \{\Phi\}$ $\quad A_5 = \{a_o\}$

$$\therefore \quad P\{A\} = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$$

$$e.c.f._{A_1}(A_o) = \text{DPCS for } A_1 = \{0,0,0\} = B_1$$

$$e.c.f._{A_2}(A_o) = \text{DPCS for } A_2 = \{1,1,0\} = B_2$$

$$e.c.f._{A_3}(A_o) = \text{DPCS for } A_3 = \{1,0,1\} = B_3$$

$$e.c.f._{A_4}(A_o) = \text{DPCS for } A_4 = \{0,1,1\} = B_4$$

$$e.c.f._{A_5}(A_o) = \text{DPCS for } A_5 = \{1,0,0\} = B_5$$

$$e.c.f._{A_6}(A_o) = \text{DPCS for } A_6 = \{0,1,0\} = B_6$$

$$e.c.f._{A_7}(A_o) = \text{DPCS for } A_7 = \{0,0,1\} = B_7$$

$$e.c.f._{A_8}(A_o) = \text{DPCS for } A_8 = \{1,1,1\} = B_8$$

Note that since the e.c.f. on A_o fixes the order of the elements of A by definition, then the e.c.f. is one-to-one and onto the set of DPCSs. Therefore, the e.c.f. inverse ($e.c.f.^{-1}$) exists and is also one-to-one and onto the power set of A. For the example above:

$$e.c.f.^{-1}\{0,0,0\} = A_1$$

$$e.c.f.^{-1}\{1,1,0\} = A_2$$

$$e.c.f.^{-1}\{1,0,1\} = A_3$$

$$e.c.f.^{-1}\{0,1,1\} = A_4$$

$$e.c.f.^{-1}\{1,0,0\} = A_5$$

$$e.c.f.^{-1}\{0,1,0\} = A_6$$

$$e.c.f.^{-1}\{0,0,1\} = A_7$$

$$e.c.f.^{-1}\{1,1,1\} = A_8.$$

Pictorically, the e.c.f., its inverse ($e.c.f.^{-1}$), and A_o can be displayed in Figure IV-1.

The concept of DP Characteristic sets (DPCSs) provides a basis for the modeling of non-numerical processes performed by digital computers. This occurs because they can represent non-binary element sets as binary element sets and therefore are compatible with digital processing. Consider the following set properties defined for DPCSs, remembering that each DPCS can be mapped back to a non-binary element DP set.

Let O_o and O_m be two DPCSs defined on the same DP set A_o. They therefore have the same power (n) and can be defined as follows:

$$O_o = \{a_1, a_5, \ldots, a_k\}$$

where

$$\bigvee_{i=1}^{n}{}_i \; a_j^i = 0 \text{ or } 1 \text{ and } a_j^i \in O_o$$

and

$$O_m = \{a_1, a_3, \ldots, a_k\}$$

where

$$\bigvee_{i=1}^{n}{}_i \; a_j^i = 0 \text{ or } 1 \text{ and } a_j^i \in O_m.$$

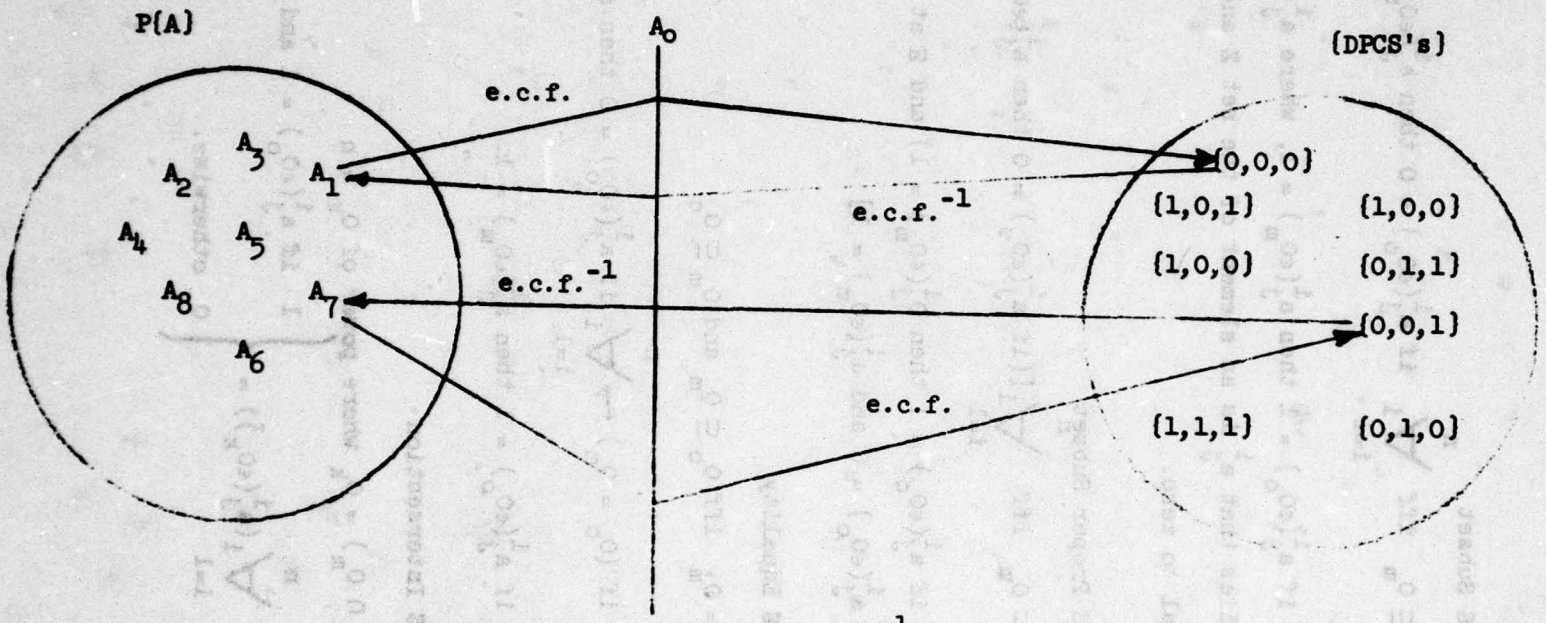


Figure IV-1. Sample e.c.f. and e.c.f.$^{-1}$.

The properties of DPCSs can now be expressed using the notation in Table IV-1.

XXI DPCS Subset

$$O_o \subseteq O_m \text{ iff } \bigvee_{i=1}^{n}{}_i \text{ if } a_j^i(\epsilon O_o) = 0 \text{ then } a_j^i(\epsilon O_m) = 0 \text{ or } 1$$

and if $a_j^i(\epsilon O_o) = 1$ then $a_j^i(\epsilon O_m) = 1$, where $a_j^i(\epsilon Z) = 0$ implies that a_j^i is an element of the set Z and it is equal to zero.

XXII DPCS Proper Subset

$$O_o \subset O_m \text{ iff } \bigvee_{i=1}^{n}{}_i [(\text{if } a_j^i(\epsilon O_o) = 0 \text{ then } a_j^i(\epsilon O_m) = 0 \text{ or } 1$$

and if $a_j^i(\epsilon O_o) = 1$ then $O_j^i(\epsilon O_m) = 1$) and $\exists$ at least one $i \ni a_j^i(\epsilon O_o) = 0$ and $a_j^i(\epsilon O_m) = 1$]

XXIII DPCS Equality

$$O_o = O_m \text{ iff } O_o \subseteq O_m \text{ and } O_m \subseteq O_o$$

$$\therefore \text{ if } (O_o = O_m) \longleftrightarrow \bigvee_{i=1}^{n}{}_i \text{ if } a_j^i(\epsilon O_o) = 0 \text{ then } a_j^i(\epsilon O_m) = 0$$

and if $a_j^i(\epsilon O_o) = 1$ then $a_j^i(\epsilon O_m) = 1$.

XXIV DPCS Intersection

$$(O_o \cap O_m) = O_k \text{ where power of } O_k = n$$

$$\text{and } \bigvee_{i=1}^{n}{}_i(a_j^i(\epsilon O_k)) = \begin{cases} 1 & \text{if } a_j^i(\epsilon O_o) = 1 \text{ and } a_j^i(\epsilon O_m) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

XXV DPCS Union

$(O_o \cup O_m) = O_k$ where power of $O_k = n$

and $\bigvee_{i=1}^{n}{}_i (a_j^i(\epsilon O_k)) = \begin{cases} 0 & \text{if } a_j^i(\epsilon O_o) = 0 \text{ and } a_j^i(\epsilon O_m) = 0 \\ 1 & \text{otherwise.} \end{cases}$

XXVI DPCS Difference

$(O_o - O_m) = O_k$ where the power of $O_k = n$

and

$$\bigvee_{i=1}^{n}{}_i (a_j^i(\epsilon O_k)) = \begin{cases} 0 \text{ if } a_j^i(\epsilon O_o) = 0 \text{ or } (a_j^i(\epsilon O_o) = 1 \\ \qquad\qquad \text{and } (a_j^i(\epsilon O_m) = 1) \\ 1 \text{ if } a_j^i(\epsilon O_o) = 1 \text{ and } a_j^i(\epsilon O_m) = 0. \end{cases}$$

The following properties of DPCS, XXVII through XXXII, are defined for completeness.

XXVII DPCS Power Set

Let X be a DPCS, then the power set of X is a set containing all its subsets, i.e.

$$P(X) = \{Y: Y \subseteq X\},$$

where Y is a DP set (not necessarily a DPCS).

XXVIII Power of a DPCS

The power of a DPCS is the number of unique elements contained in the set.

XXIX Power of a DPCS Power Set

The power of a DPCS power set is identical to property IX

and XIX, i.e. if the power of a DPCS is T then the power of its DPCS power set is 2^T.

XXX Characteristic Function of a DPCS

The domain of a characteristic function (c.f.) for a DPCS X is the power set of X. Its range is the set {0,1}. The value of c.f. is dependent upon whether or not an element contained in X is also an element of P{X}. This can be shown as:

$$\text{c.f.}: \; X \rightarrow \{0,1\} \text{ where } \text{c.f.}_{X_i}(x_j^i) = \begin{cases} 1 & \text{if } x_j^i \in X_i \\ 0 & \text{if } x_j^i \notin X_i \end{cases}$$

where $X_i \subseteq X$ and $x_j^i \in X$ (Note $x_j = 0$ or 1).

XXXI DPCS Identity Set

Define the Identity DPCS I ∋ ∃ a Normal DP set $I_k = \{1,1,\ldots,1\}$ for any $P\{I_k\} = k \geq 1$. Then $0_o \cup I_n = I_n$, $0_o \cap I_n = 0_o$ and $\forall\, n(0_o \subseteq I_n)$.

XXXII DPCS Null Set

Define the Null DPCS Φ ∋∃ a Normal DP set $\Phi_k = \{0,0,\ldots,0\}$ for any $P\{\Phi_k\} = k \geq 1$. Then $(0_o \cup \Phi_n) = 0_o$, $(0_o \cap \Phi_n) = \Phi_n$, and $\forall\, n(\Phi_n \subseteq 0_o)$.

The properties XXI through XXVI, XXXI and XXXII can be illustrated further by the following seven examples. Each example can be related back to the previous example involving DPCSs.

Example 1. $B_1 \subseteq B_2$ (or $B_1 \subset B_2$), i.e. $\{0,0,0\} \subseteq \{1,1,0\}$ but $B_2 \not\subseteq B_4$ (or $B_2 \not\subset B_4$), i.e. $\{1,1,0\} \not\subseteq \{0,1,1\}$

Example 2. $B_1 \neq B_2$ i.e. $\{0,0,0\} \neq \{1,1,0\}$ but $B_1 = B_1$ i.e. $\{0,0,0\} = \{0,0,0\}$.

Example 3. $B_3 \cap B_4 = B_7$, i.e. $\{1,0,1\} \cap \{0,1,1\} = \{0,0,1\}$

Example 4. $B_4 \cup B_5 = B_8$, i.e. $\{0,1,1\} \cup \{1,0,0\} = \{1,1,1\}$

Example 5. $B_8 - B_3 = B_6$, i.e. $\{1,1,1\} - \{1,0,1\} = \{0,1,0\}$

Example 6. B_8 = (DPCS Identity Set), i.e. $B_8 = \{1,1,1\}$

Example 7. B_1 = (DPCS Null Set), i.e. $B_1 = \{0,0,0\}$.

SUMMARY

In this chapter a description of a mathematical base for the modeling of the non-numerical functions of DBM has been presented. This base is Data Processing (DP) sets and DP Characteristic Sets (DPCS) joined with set theory and its properties. The properties of DP sets and DPCSs were defined with regard to set theory properties. Two DP set operators were presented (Push (P_s^i) and Pull (P_ℓ^i)) plus an extended property for a DP characteristic function. The chapter concluded with some examples of set theory properties for DPCSs.

Chapter V

DATA BASE MANAGEMENT AND SETS

INTRODUCTION

How well can the mathematical base described in the previous chapter model Data Base Management (DBM)? The material presented in this chapter provides an answer to this question.

The material is divided into four parts. Each part represents a level of data in DBM. These parts are:

1) the user computer interface (Reserved Word);
2) the attribute and file or relationship (F/R) names (Data Name);
3) the modifiers of the attribute and F/R names (Data Descriptors); and
4) the occurrences of the attributes and F/Rs (Data Occurrence).

To provide clarity in the discussions of these four levels, examples will be used throughout. The basic example structure shown in Fig. V-1 is from Date [12]. It defines a simple file or relation (S) and a simple job (the GET statement) to be performed against the occurrences of S. Other examples are presented where necessary.

S	S#	SNAME	STATUS	CITY
	S1	SMITH	20	LONDON
	S2	JONES	10	PARIS
	S3	BLAKE	30	PARIS
	S4	CLARK	20	LONDON
	S5	ADAMS	30	ATHENS

```
DOMAIN      S#      Character (5)
            SNAME   Character (20)
            STATUS  Numeric (3)
            CITY    Character (15)

RELATION    S (S#, SNAME, STATUS, CITY)

KEY         (S#)

GET         W(S.S.#, S.STATUS):S.CITY = 'LONDON'
```

Figure V-1. Suppliers Data Model.

USER COMPUTER INTERFACE

The user level of DBM is involved with a high level language which assists in communicating with the computer. The user communicates by forming words, numerics, expressions, instructions, etc. by putting together those alphanumeric symbols available on a key punch machine, teletype terminal, etc. These alphanumeric symbols are converted to mechanical and/or electrical codes and interpreted by the computer. The computer then performs one or more functions and responds by issuing electrical codes to a terminal, printer, etc. which can be converted to alphanumeric symbols for the user to interpret.

SYMBOLIC LANGUAGE

The symbolic language that is utilized to communicate with the computer will be modeled first. Let a set AlphaNumeric (AN) be defined as unique elements composed of those symbols that are available on a key punch machine, etc. and are interpretable by a computer. For example,

$$AN = \{A, B, \ldots, Z, 0, 1, \ldots, 9, [,], ?, \ldots, "\}.$$

A simple expression as "open file" can be modeled by a Normal DP Set (X) where $Y \in P(AN)$, i.e. $Y = \{O, P, E, N, \not b, F, I, L\}$ and $X = \{OPEN \not b FILE\} = \{O^1, P^2, E^9, N^4, E^3, F^6, L^8, I^7, \not b^5\}$. The general model for communicating at the user's level is a DP set defined on a "permutation, with replacement," of an element contained in the power set of AN.

To model the above expression at a lower level, a computer's word size can be taken into consideration. Assume the power of AN

is equal to 64 and the computer word size is six characters long. Then, as was shown in Chapter IV, the maximum number of Normal DP sets that can be created is:

$$\sum_{i=0}^{T=6} (64)^i.$$

Two of these Normal DP sets would be $X_1 = \{O,P,E,N,\not\emptyset,\not\emptyset\}$ and $X_2 = \{F,I,L,E,\not\emptyset,\not\emptyset\}$. To form the necessary expression; let

$$Y_1 = X_1 - \{\not\emptyset^6\},$$

$$Y_2 = X_2 - \{\not\emptyset^5,\not\emptyset^6\},$$

and

$$X = Y_1 \cup P_S^5(Y_2) = \{O,P,E,N,\not\emptyset,F,I,L,E\}.$$

From this example it can be seen that DP sets allow for the capability to describe character and word manipulation to describe, for example, data base structures, parsing, etc.

RESERVED WORDS

As shown above, the words that can be created are quite numerous. A small number of these words are reserved words and are keyed upon by the DBM System (DBMS). It is the utilization of these words in the proper sequences (syntax) that describes to the DBMS which job is to be performed. A method of how these words could be managed by a DBMS will be discussed and modeled using sets and DP sets.

In the example above, the word "open" can be assumed to be one of these reserved words. The words following a reserved word

or phrase are usually limited and are dictated by syntactical rules. For this same example, the DBMS would expect the next word following "open" to be a file or relationship name. The DBMS would then pass this name to the next level of processing. Before that is discussed, however, a model using DP sets will be described which attempts to give a general view of how a DBMS might utilize these reserved words and the syntactical information associated with them.

Intra-statement

The implementation of the reserved words and their modifiers can be modeled quite simply as a Normal DP set. Each element in this Normal DP set is also a Normal DP set composed of two or more elements. The first element is the reserved word followed by all of its modifiers. This can be described as shown below (without commas between elements):

{{ReservedWord/Modifier1/Modifier2/...},
{ReservedWord/Modifier1/Modifier2/...},...,
{ReservedWord/Modifier1/Modifier2/...}}.

For the example shown in Fig. V-1, the Normal DP sets may look like the following for intra-statement syntax:

IA = {...{Domain/(AttributeNames/(type(size)!)}...
{Relation/RelationName/(AttributeNames!)}...
{Get/RelationName/(RelationName·AttributeName!)}...
{:/...}...{'/Value/'}...{Key/(AttributeName)}...}.

A / has been arbitrarily chosen as a word delimitor, an exclamation point within parentheses signifies that the contents of the parentheses may be repeated, and the first word of each element in the DP set is the reserved word. These choices are arbitrary and should be viewed as an example, not a recommended procedure.

Inter-statement

The inter-statement syntactical rules are the rules between reserved words. These rules are to specify which reserved words can immediately follow a particular reserved word. This may be modeled as a Normal DP set. Each element in this Normal DP set may also be modeled as a Normal DP set where the first element is a reserved word followed by its related reserved words, e.g. {A/C/D} signifies that reserved words D and C may follow reserved word A. The format of this Normal DP set of inter-statement rules can be defined similar to the above as:

IR = {{ReservedWord/(ReservedWord!)},...,
{ReservedWord/(ReservedWord!)}}.

The number of elements in IR is equal to the number of reserved words in the user's high level language.

Utilization

After a job stream has been parsed, the first DP set (IA) would be used to detect any syntactical errors within the statement and to determine, by reserved names, which functions need to

be performed (e.g., boolean operators, write, delete, retrieve, etc.). The DBMS then determines which "words" are supposed to be attribute names or values and which are F/R names. The attribute names and F/R names would then be passed to the data dictionary and directory for further processing. The second DP set (IR) discussed above is utilized to determine if the n-th reserved word in the statements can legally follow the (n-1)st reserved word. This is accomplished by finding the list of reserved words that can legally follow the (n-1)st reserved word and determining if the n-th word or its equivalent is in the list. If so, then the DBMS knows that reserved word n can follow reserved word n-1. This same procedure would be performed for the second through the last reserved word in the user's statement sequence.

The first acceptable reserved word in a job can be modeled as a Normal DP set where its elements are DP sets with two elements. The first element contains the reserved words' name and the second element is a pointer to the location where the reserved words' entry is located in the physical implementation of IR. An example of the format of this first inter-statement Normal DP set is

FIR = {{ReservedWord/PointertoIR}, ...,
{ReservedWord/PointertoIR}}.

FIR allows the DBMS to determine if the beginning of a new job has a proper reserved word and provides a connection to IR, to be used for the second reserved word in the user's statement sequence, as discussed above.

ATTRIBUTE AND FILE/RELATIONSHIP (F/R)

Once the DBMS has determined which job(s) have to be performed, through parsing, intra-statement and inter-statement syntax checking; its next consideration is with the data. The first level of description concerned with actual data occurrences has to do with attribute and F/R names. Modeling this level of data by using sets and DP sets is considered here. This is followed by an example to illustrate what detail this mathematical base can provide to DBM.

ATTRIBUTES

Assume a data base consists of a number of attributes. Then these attributes can be modeled as a set A where the total number of attributes (TA) is the power of A and each element in A is the attribute's name $\bar{A}_i$. $\bar{A}_i$ can be modeled as a permutation, with replacement, performed on an element of P{AN}. Each attribute name, $\bar{A}_i$, can be modeled as a Normal DP set A_i. Therefore, symbolically the data base consists of the set of attributes

$$A = \{\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_{TA}\}$$

and each $\bar{A}_i$ has a Normal DP set A_i such that no two attribute names are equivalent. An example of two Normal DP set models of attribute names might be A_i = {E,M,P,L,O,Y,E,E,$\emptyset$,N,A,M,E} and A_j = {S,.,S,.,N,O}, which represent a code for social security number.

The TA attributes in the data base have unique names but some of them may have synonym attributes. Two or more attributes

are synonyms if they have different names and descriptors or different names for the same entity in the real world. An example of two synonym attribute names could be A_k = {E,M,P,L,O,Y, E,E,$\not\emptyset$,N,U,M,B,E,R} and A_j = {S,.,S,.,N,O}. These synonym attributes may occur in many ways in forming large data bases. For example, they may result from interfacing more than one current data base, adding data to existing large data bases, redefining portions of existing data bases, etc.

FILE/RELATIONSHIP (F/R)

DP sets can be used to model F/Rs as collections of attributes. An F/R for a data base can be modeled as a Normal DP set defined on a permutation, with replacement, of an element contained in P{A}. Given the set A, the set of all its subsets P{A} has a power of (See Chapter IV)

$$\sum_{i=0}^{TA} \binom{TA}{i} = 2^{TA}.$$

The elements or sets contained in P{A} are the total number of subsets of A in which F/Rs of attributes can be defined. An example of an F/R, expressed as a Normal DP set (O_i) might be

$$O_i = \{N,A,M,E,\not\emptyset,A,D,D,R,E,S,S,\not\emptyset,P,H,O,N,E,\not\emptyset,N,O\}$$
$$= A_\ell \cup P_s^4\{\not\emptyset\} \cup P_s^5 A_m \cup P_s^{12}\{\not\emptyset\} \cup P_s^{13} A_n$$

where

A_ℓ = {N,A,M,E}, A_m = {A,D,D,R,E,S,S}, and

A_n = {P,H,O,N,E,$\not\emptyset$,N,O}.

The name of an F/R may be modeled in a similar way where $\bar{O}_i$ is a permutation, with replacement, of an element contained in P{AN}. For this example O_i would be the Normal DP set

$$O_i = \{T,E,L,E,P,H,O,N,E,\not\emptyset,L,I,S,T,I,N,G\}.$$

The notion of order was utilized for modeling F/Rs because it provides a definitive structure by extending the concept of relations and it more closely resembles a computer representation. To more accurately depict a relation of two or more attributes the notion of order must be introduced. For example, let $\dot{O}_k$ = {A,N,C,E,S,T,O,R} and $\dot{O}_j$ = {D,E,C,E,N,D,E,N,T} be Normal DP sets but let A_n, A_{n+1}, O_k, and O_j be SETS where

$$A_n = \{P,A,R,E,N,T,S,\not\emptyset,N,A,M,E\},$$

$$A_{n+1} = \{C,H,I,L,D,S,\not\emptyset,N,A,M,E\},$$

$$O_k = \{A_n, A_{n+1}\}, \text{ and}$$

$$O_j = \{A_{n+1}, A_n\}.$$

Therefore, $O_k = O_j$. Note that "a parent is the ancestor of a child" is not equivalent to "a parent is the descendent of a child." Modeling the above two F/Rs using Normal DP sets generates:

$$O_k = \{\text{PARENTS}\not\emptyset\text{NAME}\not\emptyset\text{CHILDS}\not\emptyset\text{NAME}\}$$

and

$$O_j = \{\text{CHILDS}\not\emptyset\text{NAME}\not\emptyset\text{PARENTS}\not\emptyset\text{NAME}\}.$$

Modeled, using DP sets, the example illustrates that $O_k \neq O_j$, since if they were equal, then this would imply that $A_n = A_{n+1}$ and $A_{n+1} = A_n$, on a character per character basis, which is a contradiction.

EXAMPLE OF USER'S LEVEL DATA

This modeling of F/Rs and their modifiers as DP sets falls into the third level of data utilized in communicating with a DBMS. This level involves which attributes are contained in which F/R and what is the user's view of that F/R. Other models exist and were discussed earlier e.g. tree structures, networks, relational data structures, etc. The DP set model is general enough to model all of the above models as shown in Appendix A.

The fourth level of data utilized with a DBMS concerns the occurrences of the attributes and/or F/Rs. Each attribute A_i has stored in the data base one or more occurrences, designated $A_{i,j}$, where the subscript i signifies the attribute in question and the subscript j signifies which occurrence of attribute i (A_i). As an example, consider the attribute A_i being equated to {C,I,T,Y}, then a typical occurrence may be LONDON, PARIS, etc. Occurrences of attributes, as seen by the user, are modeled in the same way as O_i and A_i, i.e. $A_{i,j}$ is a Normal DP set defined on a permutation, with replacement, of an element contained in P(AN).

In concluding the discussion of the first two levels of data in DBM, an example problem is given which illustrates the DP set modeling in the static and dynamic state.

SUPPLIERS DATA MODEL

The Suppliers Data Model shown in Fig. V-1, provides an example for displaying the versatility of how sets and DP sets

can be used to model DBM at the user level. The model depicts a small data base with four different attributes which are listed under DOMAIN and one F/R called S, with a primary key (S#) used to identify unique occurrences of S. Note that all underlined words, parentheses, colons, etc. are reserved words in the language used to communicate with the DBMS. These underlined reserved words were modeled earlier in the chapter using DP sets (e.g. IA and IR).

Static State

The small data base could be constructed at the user's level using Normal DP sets in the following way:

Attribute Names (Normal DP Sets)

$$A_1 = \{S,\#\}$$

$$A_2 = \{S,N,A,M,E\}$$

$$A_3 = \{S,T,A,T,U,S\}$$

$$A_4 = \{C,I,T,Y\}$$

F/R Name (Normal DP Set)

$$\dot{O}_i = \{S\}$$

F/R Definition (Normal DP Set)

$$O_i = A_1 \cup \{\wp^3\} \cup P_s^3\{A_2\} \cup \{\wp^9\} \cup P_s^9\{A_3\} \cup \{\wp^{16}\} \cup P_s^{16}\{A_4\}.$$

$$O_i = \{S\#\wp SNAME\wp STATUS\wp CITY\}.$$

The data, or occurrences of the attributes and the F/R, can be defined in one of two ways. One way would be to define each

attribute's values first and then the proper values of each attribute could be "put together" to form the correct occurrences of the F/R. Another way would be to define the data by occurrences of the F/R.

The first way can be modeled as the following:

$A_{1,1} = \{S,1,\not{\emptyset},\not{\emptyset},\not{\emptyset}\}$,

$A_{1,2} = \{S,2,\not{\emptyset},\not{\emptyset},\not{\emptyset}\}$,

$A_{1,3} = \{S,3,\not{\emptyset},\not{\emptyset},\not{\emptyset}\}$,

$A_{1,4} = \{S,4,\not{\emptyset},\not{\emptyset},\not{\emptyset}\}$,

$A_{1,5} = \{S,5,\not{\emptyset},\not{\emptyset},\not{\emptyset}\}$,

$A_{2,1} = \{S^1,M^2,I^3,T^4,H^5,\not{\emptyset}^6,\ldots,\not{\emptyset}^{20}\}$,

$A_{2,2} = \{N^3,J^1,O^2,E^4,S^5,\not{\emptyset}^6,\ldots,\not{\emptyset}^{20}\}$,

$A_{2,3} = \{L^2,A^3,K^4,B^1,E^5,\not{\emptyset}^6,\ldots,\not{\emptyset}^{20}\}$,

$A_{2,4} = \{L^2,C^1,A^3,K^5,R^4,\not{\emptyset}^6,\ldots,\not{\emptyset}^{20}\}$,

$A_{2,5} = \{A^1,D^2,A^3,M^4,S^5,\not{\emptyset}^6,\ldots,\not{\emptyset}^{20}\}$,

$A_{3,1} = \{2,0,\not{\emptyset}\}$,

$A_{3,2} = \{1,0,\not{\emptyset}\}$,

$A_{3,3} = \{3,0,\not{\emptyset}\}$,

$A_{4,1} = \{N^3,D^4,L^1,O^2,O^5,N^6,\not{\emptyset}^7,\ldots,\not{\emptyset}^{15}\}$,

$A_{4,2} = \{P^1,A^2,R^3,I^4,S^5,\not{\emptyset}^6,\ldots,\not{\emptyset}^{15}\}$, and

$A_{4,3} = \{T^2,A^1,H^3,E^4,N^5,S^6,\not{\emptyset}^7,\ldots,\not{\emptyset}^{15}\}$.

Given the above occurrences of $(A_{i,j})$'s then the following equations define each occurrence of O_i according to the first way mentioned above:

$O_{1,1} = A_{1,1} \cup P_s^5(A_{2,1}) \cup P_s^{25}(A_{3,1}) \cup P_s^{28}(A_{4,1})$

$O_{1,2} = A_{1,2} \cup P_s^5(A_{2,2}) \cup P_s^{25}(A_{3,2}) \cup P_s^{28}(A_{4,2})$

$$O_{1,3} = A_{1,3} \cup P_s^5(A_{2,3}) \cup P_s^{25}(A_{3,3}) \cup P_s^{28}(A_{4,2})$$

$$O_{1,4} = A_{1,4} \cup P_s^5(A_{2,4}) \cup P_s^{25}(A_{3,1}) \cup P_s^{28}(A_{4,1})$$

$$O_{1,5} = A_{1,5} \cup P_s^5(A_{2,5}) \cup P_s^{25}(A_{3,3}) \cup P_s^{28}(A_{4,3}).$$

The second way requires that some of the attribute's occurrences ((A_3)'s and (A_4)'s) be duplicated since the data are being defined by occurrences of the F/R rather than the occurrences of the attributes. Therefore, the occurrences of the A_3 and A_4 attributes are redefined as:

$$A_{3,1} = \{2,0,\not\emptyset\} \qquad A_{4,1} = \{L^1,O^2,N^3,D^4,O^5,N^6,\not\emptyset^7,\ldots,\not\emptyset^{15}\}$$

$$A_{3,2} = \{1,0,\not\emptyset\} \qquad A_{4,2} = \{P^1,A^2,R^3,I^4,S^5,\not\emptyset^6,\ldots,\not\emptyset^{15}\}$$

$$A_{3,3} = \{3,0,\not\emptyset\} \qquad A_{4,3} = \{A^2,R^3,I^4,S^5,P^1,\not\emptyset^6,\ldots,\not\emptyset^{15}\}$$

$$A_{3,4} = \{2,0,\not\emptyset\} \qquad A_{4,4} = \{L^1,O^2,N^3,D^4,O^5,N^6,\not\emptyset^7,\ldots,\not\emptyset^{15}\}$$

$$A_{3,5} = \{3,0,\not\emptyset\} \qquad A_{4,5} = \{A^1,T^2,H^3,E^4,N^5,S^6,\not\emptyset^7,\ldots,\not\emptyset^{15}\}.$$

With these modifications to the data then the $(O_{1,j})$'s can be represented as:

$$\bigvee_{j=1}^{5}(O_{1,j} = A_{1,j} \cup P_s^5(A_{2,j}) \cup P_s^{25}(A_{3,j}) \cup P_s^{28}(A_{4,j})).$$

These two ways illustrate the differences encountered by defining the data for an F/R with duplicating identical occurrences of attributes and without duplicating these occurrences.

Dynamic State

To further illustrate the two definitions of this data base, consider the interpretation of the GET statement, i.e.

GET W(S.S#,S.STATUS):S.CITY = 'LONDON'.

This can be interpreted as the defining of a new F/R: $\dot{O}_2 = \{W\}$. This interpretation therefore defines $O_2 = A_1 \cup \{\not{P}^3\} \cup P_s^3(A_3)$. Obtaining the values for the attributes in O_2 is however dependent on S.CITY = 'LONDON'. The GET statement is requesting the DBMS to create an F/R named W having two attributes whose names are S# and STATUS. It is to provide occurrences for this F/R based upon the value of the attribute CITY. That is, for each occurrence in F/R, S; if the attribute CITY has an occurrence equal to LONDON then the DBMS is to retrieve the corresponding values of the occurrences of S# and STATUS and assign them an occurrence in the F/R named W. This procedure can be expressed using DP sets,

Set k = 0

$$\bigvee_{j=1}^{5}{}_j [\text{if } (O_{1,j} \cap P_s^{28}\{L^1,O^2,N^3,D^4,O^5,N^6\}) = \{L^{29},O^{30},N^{31},D^{32},O^{33},N^{34}\}$$

then for k = k + 1, set $O_{2,k} = A_{1,j} \cup P_s^5(A_{3,j})$].

If the second method for defining data were used in loading the original data then $O_{2,k} = A_{1,j} \cup P_s^5(A_{3,j})$ will not cause any problems. But, if the first method discussed were used in loading the original data, then an error would occur since $A_{3,4}$ does not exist. To correct this, $O_{2,k}$ in the above expression need to be redefined as:

$$O_{2,k} = (O_{1,j} - (O_{1,j} - \{X_{1,j}^1, X_{1,j}^2, \ldots, X_{1,j}^5\})$$

$$\cup P_\ell^{20}(O_{1,j} - (O_{1,j} - \{X_{1,j}^{28}, X_{1,j}^{27}, X_{1,j}^{26}\})))$$

where $X^{\ell}_{1,j}$ is the ℓ-th element for (O_1)'s j-th occurrence. The result of this illustration would yield the following F/R, W, as shown by Date [12]

W	S#	STATUS
	S1	20
	S4	20

MODIFIERS OF ATTRIBUTES AND FILE/RELATIONSHIPS

The next level of modeling concerns the DBMS's overhead data for insuring that the data descriptions are utilized correctly and that the correct occurrences are retrieved. This function can be performed through the utilization of a data dictionary and a data directory. They will be described in detail, related to the sample problem and modeled using sets and DP sets.

DATA DICTIONARY

A data dictionary provides data about the attributes in the data base. These data are needed by the DBMS for the eventual processing of the attribute's occurrences. An example of the kind of data that may be contained in a data dictionary are:

1) representation,
2) format,
3) size, and
4) synonym data.

Once a job has been received by a DBMS, parsed, and checked for

syntax errors the DBMS then accesses these data contained in the data dictionary.

REPRESENTATION

The representation technique for an attribute's occurrence determines how the computer will represent and treat it in binary form. The modeling of a data dictionary concerning a representation reverts back to the definition of the set A. This set can be partitioned into t mutually exclusive sets depending on what the user has defined to the computer as the representation technique required for the occurrences of each of the attributes. This variable t is computer dependent and represents the power of the set, X, of different representation techniques available on a computer, i.e., X = {f.p.,i,b,a*l*,etc.}, where f.p. represents floating point, i represents integer, b represents boolean, and a*l* represents alphanumeric. If, $\bar{A}_1$, $\bar{A}_5$, ..., $\bar{A}_x$ are attributes whose occurrences are represented as floating point then let

$$F.P. = \{\bar{A}_1, \bar{A}_5, \ldots, \bar{A}_x\}.$$

Similarly, the set A can be partitioned into sets I, B, and AL, where I, B, AL, and F.P. are mutually exclusive. This implies that no attribute's occurrences can be represented in more than one way and therefore A = {F.P. U AL U B U I}.

Format and Size

Within each of these subsets of A discussed above, other subsets can be formed. These subsets are mutually exclusive and

are based on the definition of the format and/or the maximum size of the occurrences of each attribute, $\bar{A}_i$. If the subset is AL then it can also be partitioned into subsets. The criteria for this partitioning can be the maximum size of the attribute's occurrences, e.g. subsets of one character long (SAL1), two characters long (SAL2), etc. If, however, the subset of A is F.P., then it can be partitioned in two different ways; by equivalent formats and size. Partitioning by equivalent formats implies for example that all attributes whose occurrences have a floating point format of F7.3 would be defined as one subset and all attributes whose occurrences have a floating point format of F7.2 would be defined as another subset, thus

$$F7.3 = \{\bar{A}_2, \bar{A}_{20}, \ldots, \bar{A}_\ell\}$$

and

$$F7.2 = \{\bar{A}_1, \bar{A}_{19}, \ldots, \bar{A}_k\}.$$

This example implies, for instance, that $\bar{A}_{20}$ is represented as a floating point attribute where there are 7 characters reserved for its occurrence, there are 3 characters reserved for those numbers right of the decimal point and 4 (7 - 3 = 4) characters reserved for those numbers left of the decimal point. Note that $(F7.3) \cap (F7.2) = \Phi$ and similarly $(SAL1) \cap (SAL2) = \Phi$ which implies that no one attribute can have its occurrences defined with two different formats or sizes.

The other way in which the subset F.P. can be partitioned is by size. This can be accomplished similarly as for the subset AL; i.e. F.P. can be divided into subsets of maximum

sizes of two characters long (S F.P.2), etc. For the above example the subset of seven characters long would be

$$((F7.3) \cup (F7.2) \cup (F7.0) \cup (F7.1) \cup (F7.4) \cup (F7.5) \cup (F7.6)) = \{\bar{A}_2, \bar{A}_{20}, \ldots, \bar{A}_{\ell}, \bar{A}_1, \bar{A}_{19}, \ldots, \bar{A}_k, \ldots\}.$$

Note that for these examples, size was measured in characters; the same could have been accomplished by measuring the size in bits.

Synonyms

Two or more attributes are synonyms if they have different names and descriptors or different names for the same entity in the real world. In the sharing of large data bases, there are certain attributes used by different jobs that necessitate having the same attribute's occurrences defined differently. This requirement is one of the reasons for having synonym attributes in a data base. Since synonym attributes have different names, then they may have a different representation and/or a different format, and/or a different size. An example of this might be the following: $\bar{A}_i$ = S.S.# and $\bar{A}_k$ = S.S.No. where $\bar{A}_k$ may be represented as an integer and $\bar{A}_i$ may be represented as an Alphnumeric. One occurrence of $\bar{A}_i$ and $\bar{A}_k$ might be modeled using Normal DP sets as

$$A_{i,5} = \{0,0,1,4,7,8,2,0,8\}$$
$$A_{k,5} = \{1,4,7,8,2,0,8\}$$

where the actual employee's social security number appearing on his/her employment card would be "001-47-8208."

Dictionary Construction

The actual construction of a data dictionary is problem dependent. For instance, if the attribute's occurrences have maximum fixed sizes, then these values will probably appear in the dictionary since they are constant modifiers or size descriptor values. However, if the size descriptor of an attribute's occurrences are variable, i.e., are different for each occurrence of an attribute, then the value for the size descriptor will probably not appear in the data dictionary but will appear in the data directory. This is so since it modifies the occurrences as they are stored on the computer more than the general definition of an attribute's occurrences. If, for instance, a data base only has one representation and format descriptor (e.g. aℓ) then there would be no need to include the value for a representation for each attribute in the data dictionary.

The last entry that may appear in a data dictionary is related to descriptions, functions, and synonyms. If a data base has synonyms then there are two major ways in which the synonyms can be handled:

1) (Type I synonyms), store the occurrences of the synonyms in only one description and develop additional functions to convert the different occurrences from one description to another; and

2) (Type II synonyms), store the occurrences of the synonyms in its different descriptions.

There are, however, advantages and disadvantages to both of the above approaches. For instance, the first approach enlarges the data dictionary and requires more processing just to retrieve information. The second approach increases the size of the data base and causes a problem with data integrity in that the same occurrences are stored multiple times in the computer under different attribute names. The functions required by the first approach are all different because their domains and ranges are different. The domain and range might be the sets (F9.2) and (SI3) (Integer, 3 characters long) where (F9.2) might be the format for $\bar{A}_1$ = ANNUAL SALARY and (SI3) might be the format for $\bar{A}_j$ = NEAREST K$ SALARY. Another example might be $\bar{A}_i$ = S.S.# and $\bar{A}_j$ = S.S.No. where the range and domain could be (SI8) and (SAL8), respectively.

Dictionary Model

The modeling of a data dictionary requires at least a unique and shortened Normal DP set for each attribute name ($\bar{A}_j$) and F/R ($\dot{O}_i$). The rest of what is contained in a data dictionary is problem dependent and could contain some, all, or more than the elements in the following DP set model of the i-th entry of a data dictionary:

$$DIC_i = \{AC_j \cup P_s^x(AN_j) \cup P_s^{x+y}(CD) \cup P_s^{x+y+z}(S_t)$$

$$\cup P_s^{w+x+y+z}(AC_p) \cup P_s^{2x+y+z+w}(H_r)\},$$

where AC_j - Normal DP set of the attribute's shortened name,

AN_j - Normal DP set of the attribute's name,

CD - Normal DP set of the representation technique descriptor,

S_t - Normal DP set of the size descriptor,

AC_p - Normal DP set of the attribute's synonym shortened name,

H_r - Normal DP set of the name of the function to convert the occurrences of the synonyms, and

x, y, z, and w are the powers of the Normal DP sets AC_j (and AC_p), AN_j, CD, and S_t, respectively.

This model refers to the i-th occurrence in the data dictionary containing data about an attribute in the data base. If, however, the i-th element in the data dictionary contains data about an F/R, then a model using Normal DP sets could be the following:

$$DIC_i = \{AC_j \cup P_s^x(AN_j) \cup P_s^{x+y}(S_t) \cup P_s^{x+y+w}(AC_p)\}$$

where AC_j - Normal DP set of the (F/R)'s shortened name,

AN_j - Normal DP set of the (F/R)'s name,

S_t - Normal DP set of a pointer to the (F/R)'s hashing algorithm and/or directory, and

AC_p - Normal DP set of the primary keys shortened name.

Dictionary Example

As an example, consider the Suppliers Data Model from Date [12], where the data dictionary could be modeled as

$DIC_1 = \{1,/,S,\#,/,a,\ell,/,5\}$

$DIC_2 = \{2,/,S,N,A,M,E,/,a,\ell,/,2,0\}$

$DIC_3 = \{3,/,S,T,A,T,U,S,/,i,/,3\}$

$DIC_4 = \{4,/,C,I,T,Y,/,a,\ell,/,1,5\}$

$DIC_5 = \{5,/,S,/,X,X,/,1\}$

where / (slashes) are used as delimiters 1, 2, 3, 4, and 5 are used for shortened names for S#, SNAME, STATUS, CITY and S, respectively; and XX is the pointer to S's directory or hashing algorithm. Note that DIC_1 through DIC_4 pertain to attributes while DIC_5 pertains to the relation S.

Continuing with Date's example, consider the GET statement, i.e. GET W(S.S.#,S.STATUS): S.CITY = 'LONDON'. To process this job the DBMS must first operate on the data dictionary to find the descriptors of each of the attributes in question (i.e. S#, STATUS, and CITY). Therefore, the DBMS searches the data dictionary until it finds the proper attribute then writes the proper DIC_i into the user's area of the computer memory for further processing. An example of this process using DP sets can be shown for the attribute STATUS as follows:

Set i = 1

1) If $\{S,T,A,T,U,S\} \cap (P^2_\ell(DIC_i)) = \{S,T,A,T,U,S\}$

   Go To 2

   i = i + 1 Go To 1

2) Write DIC_i into user's area.

Once the desired attributes are found (or it is determined that they do not exist in which case processing will be stopped) the proper F/R's can be found in a similar way.

In summary, attributes' names and F/R's names may be stored in a data dictionary. The descriptors that may be found for any attribute name are the attribute's shortened name, representation technique, format, size, and synonym data. The descriptors that may be found for an F/R name are the F/R's shortened name, a pointer to its directory and/or hashing algorithm and its primary key's shortened name.

The elements in the data dictionary relating to the attributes, provide the DBMS the information required to process their occurrences when they are written into the user's area of the computer memory. The method of obtaining these occurrences is dependent on the F/R element in the data dictionary. For the example above, the two important elements are the pointer to the (F/R)'s hashing algorithm and/or directory and the primary key's shortened name. The pointer may point to either a data directory or a hashing algorithm and directory. If the pointer points to the data directory, then it may be assumed that the F/R occurrences will follow contiguously, through pointers, or through an inverted list based upon the primary key's shortened name. Otherwise the directory can be assumed to be stored with the hashing algorithm also based upon the primary key's shortened name.

DATA DIRECTORY

A data directory can be thought of as a data base road map. It directs the computer to the required occurrences of data once the data dictionary has been processed. A data directory is modeled in this portion of the chapter using sets and DP sets. This is followed by an example. A discussion of data retrieval techniques is provided along with models of an inverted list and a disk device.

Directory Model

A data directory can be modeled as a Normal DP set. Again, like the data dictionary, the actual structure of a data directory is data base or problem dependent. To show how a data directory can be modeled by DP sets consider the following general model for a tree data structure. The first element is an integer (Ix_1) depicting the size of the following element, the second element is the shortened name of the $F/R(\dot{O}_i)$, the third element is Ix_2, an integer depicting the number of attributes in $\dot{O}_i$, the fourth element is an integer (Ix_3) depicting the number of occurrences of the relationship (if Ix_3 is greater than 1, then this implies a "repeating group"), the fifth element is an integer (Ix_4) designating the number of subrelations ("subsumed relations"), and the sixth element is an integer (Ix_5) designating the size of the following element. The sixth element is similar to the first element. The seventh element will be the first attribute's name A_i in the relation. The sixth and seventh elements will repeat

pairwise for Ix_2 times. Then, there will be a list of Ix_3 zeros separated by delimiters, which are "place holders" for pointers to other locations in the computer. These locations will contain the occurrences $O_{i,j}$ of the relationship desired. Then the above starts over again for at least the number of subrelations Ix_4. The reason why it is <u>at least</u> Ix_4 times is because a subrelation may have subrelations.

<u>Directory Example</u>

Consider the following Normal DP set model for Date's example where "//" signifies delimiters before and after a data directory and "/" signifies a delimiter between elements.

DIR = {/,/,1,/,S,/,4,/,1,/,0,/,2,/,S,#,/,5,/,S,N,A,M,E,/,
6,/,S,T,A,T,U,S,/,4,/,C,I,T,Y,/,V,/,X,/,Y,/,Z,/,/}.

The values of V, X, Y, and Z are the computer locations of the 4 occurrences of S. In location V the following Normal DP set model would depict the first occurrence of F/R S:

S_1 = {/,/,2,/,S,1,/,5,/,S,M,I,T,H,/,2,/,2,0,/,
6,/,L,0,N,D,0,N,/,P,1,/,/}

where P1 is the location where X is stored in the DIR model.

Considering the same example used before, i.e. <u>GET</u> W (S.S#,S.STATUS): S.CITY = 'LONDON', the pointer in the data dictionary has located DIR shown above. Before the DBMS begins to process the occurrences of S, it must first evaluate DIR to determine if, in fact, it does represent S and if the attributes

in question (i.e. S#, STATUS, and CITY) are in fact attributes in S. The DBMS would evaluate the contents of DIR through a software code whose logic would be directed by the elements in DIR. Consider the following brief example as a model of what this code is required to do, for first verifying that the F/R directory is for S and whether or not the first attribute is S#.

If $\{S\} \cap P_{\ell}^{4}\{DIR\} = \{S\}$ CONTINUE

$\vdots$

If $\{S,\#\} \cap P_{\ell}^{14}\{DIR\} = \{S,\#\}$ CONTINUE.

(These examples are very simple and are presented only to show in detail that DP sets can be used in describing non-numerical processing in DBM.) Once all the attributes in question have been found in DIR then the DBMS may proceed in following the pointers (V,X,Y,Z) to find the occurrences of S and possibly writing them into the user's work area for the data processing required to provide the proper result (W).

Directory/Data Retrieval

There exists data retrieval techniques other than the one discussed above. The example depicts what may occur if the occurrences were stored and found through the utilization of pointers imbedded in DIR. If the occurrences were stored sequentially then pointers would not have been needed and the occurrences would have been contiguous with DIR. Other techniques could also be modeled such as binary trees, single linked lists, double linked lists, etc. However, if the occurrences are stored

by a hashing algorithm then the pointer in the data dictionary would point to DIR and following DIR would exist a mathematical algorithm that would hash on the primary key value and generate an address for the proper occurrence of the F/R required. Hashing algorithms are numerical in nature and therefore no attempt has been made to model them using DP sets.

Another major way in which the occurrences may have been located is through the use of inverted lists. These can be modeled using DP sets. Let an inverted list be modeled as a DP set where one element can be modeled as the following Normal DP set:

$$\{\{A_i\}, \{A_{i,1}, L_i, L_j, \ldots, L_p\}, \{A_{i,2}, L_a, L_s, \ldots, L_{r1}\},$$

$$\ldots, \{A_{i,p}, L_i, L_d, \ldots, L_r\}\},$$

where A_i is a Normal DP set of the primary or secondary key's name,

$A_{i,j}$ is a Normal DP set of the primary or secondary key's occurrence, and

L_x is a Normal DP set of the primary or secondary key's occurrence's address or location (i.e., a pointer).

Note for primary key attributes there will only exist one L_x per $A_{i,j}$. A fully inverted list will contain as many of the above elements as there are attributes in the F/R.

In the inverted list above, if the occurrences are stored in main memory, then these pointers (L_x) will contain their address or memory location. If these occurrences are located, say on a

disk or drum, then these pointers would designate the portion of the peripheral device that should be copied into main memory. An example of a pointer to a disk may be constructed as a Normal DP set where the first element is the disk name D_i, the second element might be an integer denoting the cylinder I_k, the third element might represent the track T_j, and the last element might represent the block B_ℓ. Given this description, the format for a pointer to a physical block would be

$$\{D_i, I_k, T_j, B_\ell\}.$$

In summary, a data directory was discussed and modeled using DP sets. An example was presented for illustrative purposes. The data directory assists in providing to the user's work area the occurrences of the data requested. Once the data are in the user's work area then they can be processed by the DBMS's codes.

OCCURRENCES OF ATTRIBUTES AND FILE/RELATIONSHIPS

The occurrences of the data that the computer operates upon are in a binary code. The modeling described in this chapter has been performing on elements contained in the character set AN. It is the purpose of this portion of the chapter to present the logic that shows that there exists a one-to-one mapping between the modeling performed at the character and the modeling performed at the bit level. Accomplishing this will show that sets and DP sets can model the non-numerical processing of DBM for all four levels of data. The methodology used is to show a one-to-one

relationship for one character. Since each character's binary representation is unique, then the joining of more than one character also produces a unique and one-to-one correspondence to its binary representation.

ALPHANUMERIC/BINARY

The data that the user describes to the DBMS must be transformed to a binary abstraction compatible with the digital computer. This binary abstraction does not, however, destroy any of the models developed so far. This is so since all communication with the computer was through elements of P{AN} and every unique element in AN can be modeled as a unique Normal DP set of power 1 (e.g. $\{A^1\}$, $\{B^1\}$, etc.). Examples of this are such codes as ASCII standard code (8 bit). From Foster [16] a one-to-one correspondence of the unique elements in AN and their uniquely ordered elements can be seen.

These uniquely ordered elements (K) are all either zero or one. These can be modeled as a Normal DP set B_i defined on a permutation, with replacement, of an element contained in P{0,1}. In the case of ASCII the number of zeros and ones is 8. This implies that there are 2^8 or 256 unique elements (Normal DP sets) that can be obtained from P{0,1}, i.e.

$$(P\{0,1\})^k = 2^8 = 256.$$

Let this set of 256 elements (B_i's) be called the total binary set or TB. It is now possible to define a function g which maps

each element in AN to its respective element in TB. An example of the above defined terms and modeling can be seen as follows:

AN	g(AN) = $B_i \in$ TB
∅	$\{1^1,0^2,1^3,0^4,0^5,0^6,0^7,0^8\}$
!	{0,0,1,0,0,0,0,1}
+	$\{0^1,1^8,0^2,1^7,1^3,0^6,0^4,1^5\}$
H	{0,1,0,0,1,0,0,0}

The function g is between sets AN and TB such that:

1) the Domain of g, D(g), is equal to AN, and

2) if g(x) = y and g(x) = z, then y = z where x ∈ AN and z ∈ TB.

It should be noted that since the power of TB is 256 and the power of AN is less than 256, then g is defined as being one-to-one and into. A function, f, is said to be a function from A into B if the range of f is not equal to B or f is not onto B. This implies that the Range of f is a proper subset of B. For the above example since g is one-to-one and into this implies that there exists an element (B_i) in TB such that there is an element (a_i) in AN such that $g(a_i) \neq B_i$. This leads to the definition of the subset of TB which is not in the Range of g, say NTB, such that

$$NTB = (TB - \bigvee_{i=1}^{PAN}{}_i(g(a_i)))$$

where $a_i \in$ AN and PAN = power of AN.

It is necessary to continue with the above modeling philosophy in order to define a function from TB to AN. This provides part of the modeling portion for the communication of the computer back to the terminal, or printer, etc. But, the power of TB is greater than the power of AN, therefore an artificial variable τ will be defined as an element of AN. In actuality, this variable, τ, could represent those cases of errors caused by interference in data transmission.

Given the above, a function h, similar to g, may be defined for TB to AN as:

1) the domain of h, D(h), is equal to TB;

2) if $h(y) = x$ and $h(y) = z$, then $x = z$;

3) if $y \in NTB$, then $h(y) = \tau$; and

4) if $y \in (TB - NTB)$, then h is equal to the inverse of g.

These two functions g and h have closed the modeling loop between the user's high level language and the digital computer's language (binary) and back again for one character. It can be easily seen that a Normal DP set defined on a permutation, with replacement, of two or more elements in P{AN}, also has a one-to-one mapping to TB and back again. Consider the following example.

Let $A_i = \{H,+\} = a_j \cup P_s^1(a_k)$, where $a_j = \{H\}$ and $a_k = \{+\}$, then $g(A_i) = g(a_j) \cup P_s^8(g(a_k)) = \{0,1,0,0,1,0,0,0,0,0,1,0,1,0,1,1,\}$. For $h(\{0,1,0,0,1,0,0,0,0,0,1,0,1,0,1,1,\})$, or the return of A_i, let

$$X = \{0,1,0,0,1,0,0,0,0,0,1,0,1,0,1,1,\}.$$

Let $Y = (X - Y_1)$ where $Y_1 = \{0^9,0^{10},1^{11},0^{12},1^{13},0^{14},1^{15},1^{16}\}$ and

$$Y_2 = P_{\ell}^{8}(Y_1)$$

then $(A_i) = h(Y) \cup P_s^1 h(Y_2) = \{H,+\}$.

SUMMARY

This chapter's purpose was to provide a description of how the mathematical base provided in Chapter IV could model DBM. The chapter was divided into four parts covering a range from that level seen and utilized by a user down to the bit level of a digital computer. Throughout the chapter, where appropriate, examples were used to clarify the concepts presented.

The next chapter contains a description of a proposed hardware implementation of a DP characteristic set model. The model describes the functions performed within a data dictionary and partial data directory by a DBMS.

Chapter VI

DICTIONARY/DIRECTORY PROCESSOR (DDP)

INTRODUCTION

Four levels of data exist and can be seen by a user through interacting with a DBM system. These levels can be called:

1) Reserved Word,

2) Data Name,

3) Data Descriptor, and

4) Data Occurrence.

The Reserved Word level includes those words that are "reserved" for the DBM system, e.g. the words GET, KEY, RELATION, and DOMAIN, as shown in Fig. V-1. The Data Name level includes those names specified by the user to identify specific attributes and F/Rs in the data base. Examples of these names, as shown in Fig. V-1, are the F/R name S and the attribute names S#, SNAME, STATUS, and CITY. The Data Descriptor level includes those words and symbols that are utilized to "describe" the occurrences of the names in the Data Name level. Referring to Fig. V-1 these are:

1) the words Character and Numeric;

2) the domain sizes designated as 5, 20, 3, and 15; and

3) the F/R name S, and its description (S#, SNAME, STATUS, CITY).

The Data Occurrence level includes those data most often sought

by the user. Referring to Fig. V-1, note the attribute names and F/R name S and their occurrences:

Name	An Occurrence
S#	S_4
SNAME	BLAKE
STATUS	10
CITY	LONDON
S	(S_2, JONES, 10, PARIS).

Associated with each of the above levels are inherent functions performed by a DBMS. For instance, at the Reserved Word level, the DBMS parses the data and performs syntactical functions. At the Data Name level the DBMS must build and maintain a data dictionary and data directory. The Data Descriptor level provides the data for the data dictionary and directory. At the Data Occurrence level the DBMS performs equal to, less than, and greater than search functions, sort functions, add functions, and delete functions.

In previous work some of the functions associated with the Data Occurrence level have been implemented in hardware. These were described in Chapter II. They included the use of associative processors, logic on disk devices and hypothetical machines.

This research is concerned with some of the functions associated with the Data Name and Data Descriptor levels and their implementation in hardware. These functions operate on a DBMS's data dictionary and data directory. A data dictionary

can be thought of as that part of a DBMS where information is maintained about the characteristics of the data occurrences. A data directory is that portion of a DBMS that maintains information pertaining to the relationships, both physical and logical, between data names and their occurrences. The data names and descriptors that have been chosen to illustrate a hardware implementation of a data dictionary and partial directory processor, or Dictionary/ Directory Processor (DDP) are the following:

1) Attribute Name - The name of an attribute as known by the user of the DBMS.
2) Attribute's Shortened Name - A shortened attribute name utilized for computing efficiency.
3) Representation - The name of the representation technique for an attribute's occurrence.
4) Size Descriptor - The size descriptor for an attribute's occurrence.
5) Synonym Function - The name of the function that converts an attribute's occurrences to a format compatible (both in representation technique and size) with its synonym's occurrences.
6) Uniqueness Descriptor - An attribute descriptor for designating whether an attribute's occurrences are unique.
7) Password Name - A name used by the DBMS to aid in restricting access to an attribute's occurrences.
8) Privileged Name - A name used by the DBMS to aid in

maintaining the validity of an attribute's occurrences.

9) F/R Name - The name of an F/R as known by the user of the DBMS.

10) F/R Shortened Name - A shortened F/R name utilized for computing efficiency.

11) F/R Primary Key - An attribute's shortened name of the key for an F/R.

12) F/R Pointer - A pointer (directly or indirectly) to an F/R's inverted list, hashing algorithm, or data directory.

13) Containment Map - A descriptor which designates which attributes are associated with each F/R.

14) Bit Map - A descriptor that designates which F/Rs are related to each other.

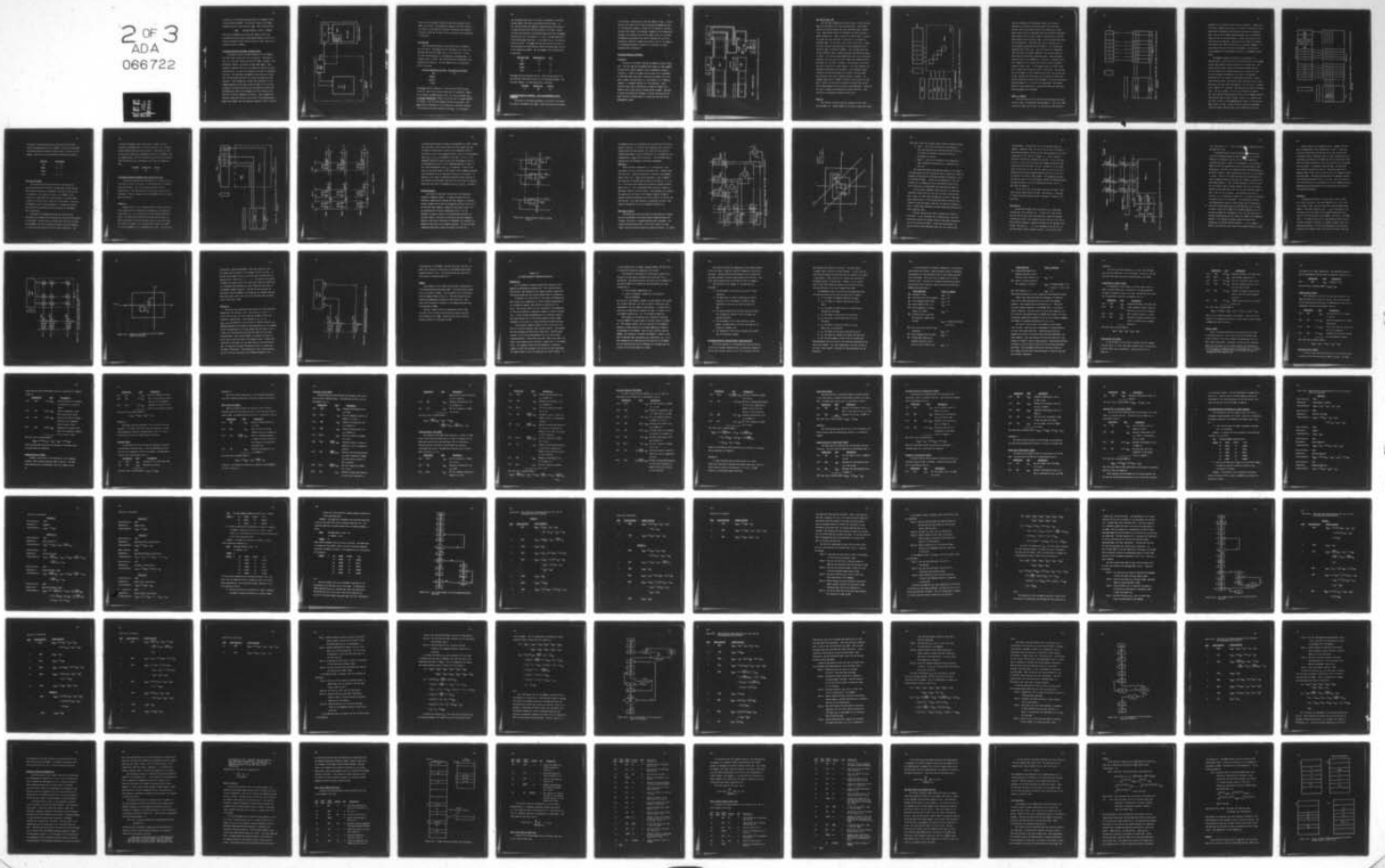
SAMPLE PROBLEM

The sample problem shown in Fig. V-1 is concerned with one F/R (S), four attributes (S#, SNAME, STATUS, and CITY) and a key (S#). The shortened names for S, S#, SNAME, STATUS, and CITY were chosen as 5, 1, 2, 3, and 4, respectively. The form of representation for S#, SNAME, and CITY is Alphanumeric or Character (aℓ) and for STATUS, integer or numeric (i). The size descriptors for S#, SNAME, STATUS, and CITY are 5, 20, 3, and 15 characters. The F/R pointer has been chosen as XX. The problem is to provide to the user an F/R, W, based on the

AD-A066 722 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB N Y F/G 9/2
A DATA BASE MANAGEMENT MODELING TECHNIQUE AND SPECIAL FUNCTION --ETC(U)
JAN 79 G T CAPRARO, P B BERRA

UNCLASSIFIED RADC-TR-79-14 NL

2 OF 3
ADA
066722



occurrences of the attribute CITY; where W is composed of the attributes S# and STATUS. The problem is posed to the DBMS through one of its reserved words, GET. This is expressed as:

GET W(S.S#,S.STATUS): S.CITY = 'LONDON'.

This can be interpreted as asking the DBMS to provide the occurrences of the attribute names S# and STATUS, of F/R S, for those occurrences of F/R S whose attribute's name, CITY, has an occurrence equal to LONDON.

DICTIONARY/DIRECTORY PROCESSOR INTERFACE LEVEL

The method by which the DDP interfaces with the DBMS to solve the above problem will now be described. Referring to Fig. VI-1, the user communicates with the DBMS, resident on the sequential computer, and submits his/her GET command. The sequential computer directed by the DBMS will perform its overhead functions, e.g. parse the command and perform its syntactic functions. This provides the DBMS with the necessary information for performing functions on the data dictionary and data directory. This information includes which characters in the command represent an F/R name, attribute name, or an attribute occurrence, and which attributes are associated with which F/R. The DBMS would direct the arrangement of this information and data into a form of commands and controls. These commands and controls, along with the data, are stored in the transfer/user's memory area (TUMA), and the sequential computer's CPU is notified

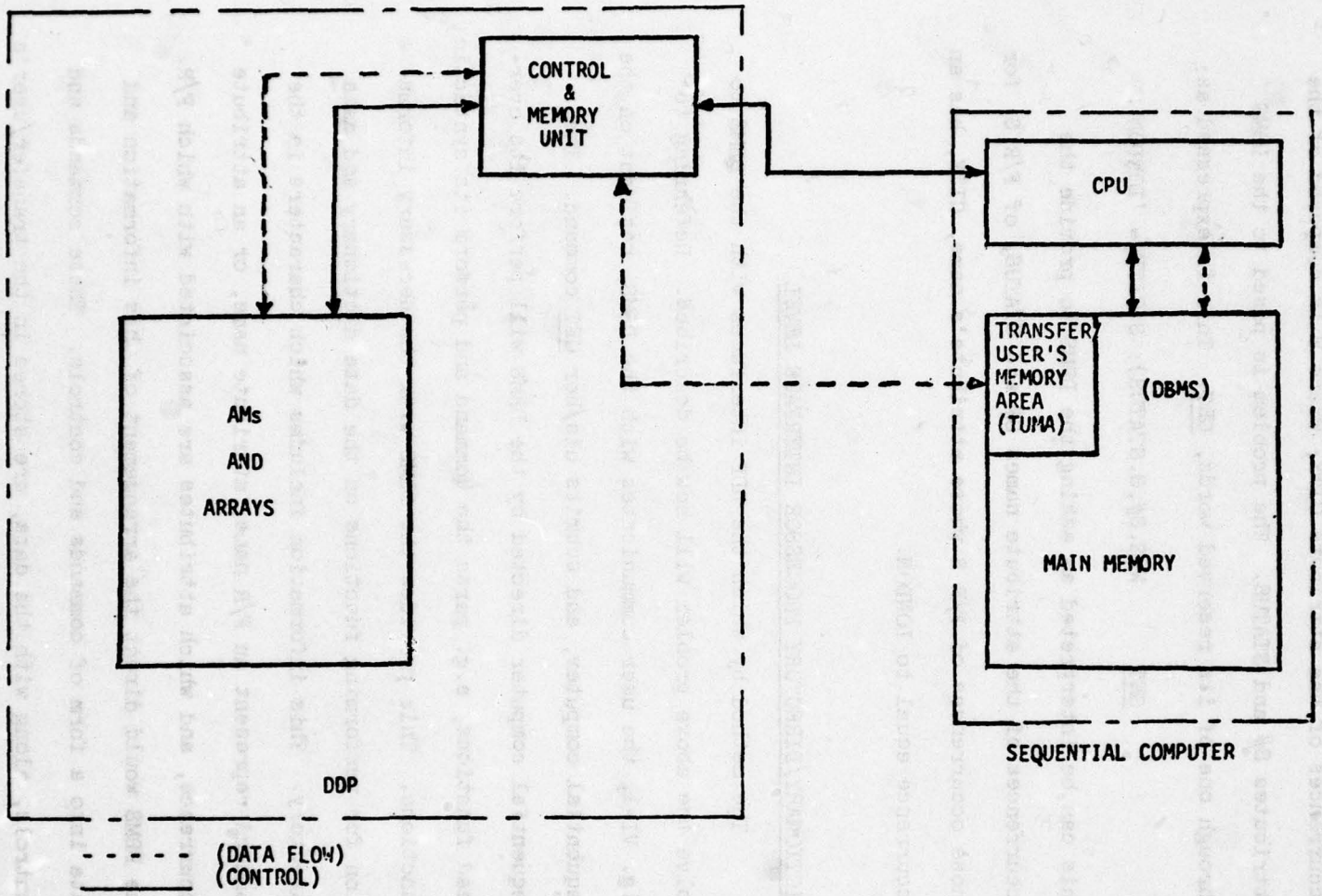


Figure VI-1. Dictionary/Directory Processor - Sequential Computer Interface.

that a job for the DDP is ready for execution and where in the TUMA it is located. The sequential computer's CPU then notifies the control of the DDP about the above information and assigns a location in TUMA for the DDP to store its results after completing its functions.

Six Functions

The functions performed by the DDP provide six different sets of results. The first function determines if, in fact, the F/R names and attribute names exist in the data base. If they do, then the DDP provides to the DBMS their shortened names. If a name(s) does not exist, then the DDP informs the DBMS and stops processing the function. For the example above, the following would be provided:

Attribute Name/Shortened Name	F/R Name/Shortened Name
S#/1	S/5
SNAME/2	
STATUS/3	
CITY/4	

The second function determines if the attributes have synonyms. If they do, then their attribute names must be determined, depending on whether the DBMS supports type I or type II synonyms (see DICTIONARY CONSTRUCTION, Chapter V) and the type of command making the request. For the above example there are no synonyms. The third function determines if the attributes in the user's request are associated with their respective F/R name. In the example,

the containment map would be utilized to determine if the F/R S has S#, STATUS, and CITY as associated attribute names. If not, the DDP informs the DBMS and stops processing the function. If there were more than one F/R involved in the user's request such that they implied a relationship among them; then the fourth function would be to determine if, in fact, the DBMS supported a relationship among these F/Rs. If the DBMS did not, the function would be aborted. The fifth function would provide the rest of the descriptors for each attribute name and synonym name involved in the requesting command. For the example, the following would be provided:

Attribute Name	Representation	Size
S#	al	5
SNAME	al	20
STATUS	i	3
CITY	al	15

The sixth function provides the F/R primary key and pointer for each F/R and related F/R involved in the requesting command. For the above example, the following would be provided:

F/R Name	Primary Key	Pointer
S	S#	XX

DICTIONARY/DIRECTORY PROCESSOR - DATA BASE MANAGEMENT SYSTEM INTERFACE

The above six functions performed by the DDP are controlled by a Control and Memory Unit (CMU). These functions are performed

by utilizing a configuration of AMs and ARRAYS of logic. Control and data are passed from the CMU to the AMs and ARRAYS and then to the sequential computer. Once a job is completed by the DDP, the DDP's CMU informs the sequential computer of its findings and transfers the resultant data to the proper location in the TUMA. The DBMS can then proceed with either informing the user of an incorrectly formulated command or proceed to fulfill the command by retrieving the proper occurrences of the F/R(s) and their remaining data directory(s).

DICTIONARY/DIRECTORY PROCESSOR

AMs/ARRAYS

The heart of the DDP is the AMs and ARRAYS as shown in Fig. VI-2. The four AMs and one Random Access Memory (or AM), RAM/AM, are physically linked together by four programmable arrays of "switches." A major and common function that will be performed on these AMs many times over is searching its memory for a word, whose contents are equivalent to a word placed in its comparand register. A secondary function is a link function. This function causes one or more words in an AM or an ARRAY to be linked to one or more words in another AM or an ARRAY. The word or words controlling this linkage is usually determined by the major function discussed above in conjunction with one of the programmable arrays.

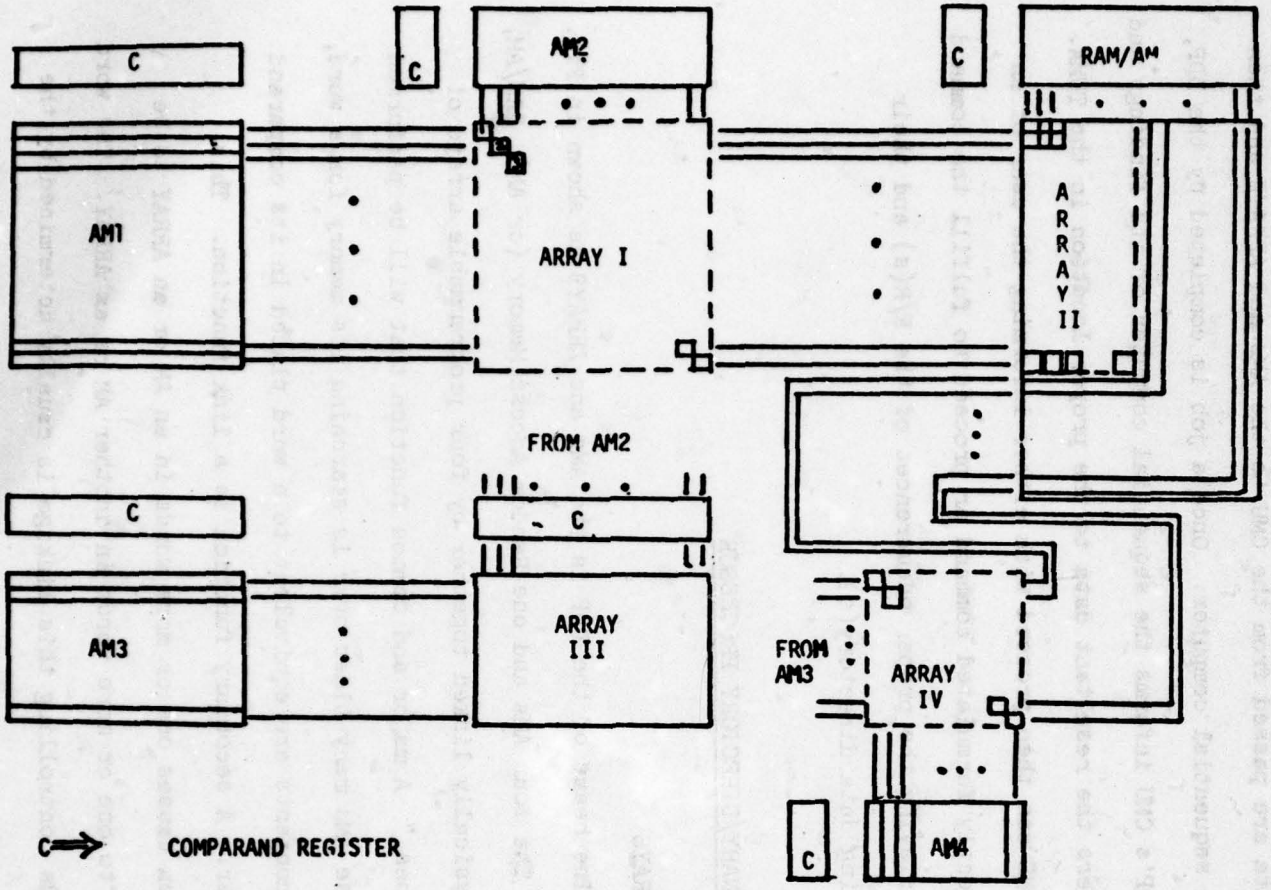


Figure VI-2. Dictionary/Directory Processor Memory and Array Architecture.

AM1, AM2 and AM3, AM4

The four AMs, RAM/AM and the three arrays, contain the data names and descriptors that were discussed above. AM1 and AM2 contain the Attribute Names and their shortened names, respectively. AM3 and AM4 contain the F/R Names and their shortened names, respectively. AM1 is connected to AM2 and AM3 is connected to AM4 such that each name is connected to its respective shortened name. To illustrate this, refer to Fig. VI-3. Assume the first function discussed above is being performed and the attribute name in question is STATUS. Then the CMU would load the word STATUS in AM1's comparand register and perform an equal-to-search. The attribute name STATUS does exist in the data base. Therefore this will cause the word in AM2, where STATUS's shortened name (3) is stored, to respond. The contents of this word can then be read into the CMU's memory. This same procedure can similarly be performed for F/R names to their shortened names. Inversely, F/R shortened names and attribute shortened names once found can cause their respective names to respond. The diagonal square cells in ARRAY I contain a flip flop and two AND gates. These cells provide the connection between AM1 and AM2 which helps perform the second function discussed above. Similarly, the cells in ARRAY IV help perform the fourth function. These cells will be discussed in greater detail later.

ARRAY III

The contents of AM2 and AM3 are connected to each other through ARRAY III. Imagine ARRAY III as being a large matrix where

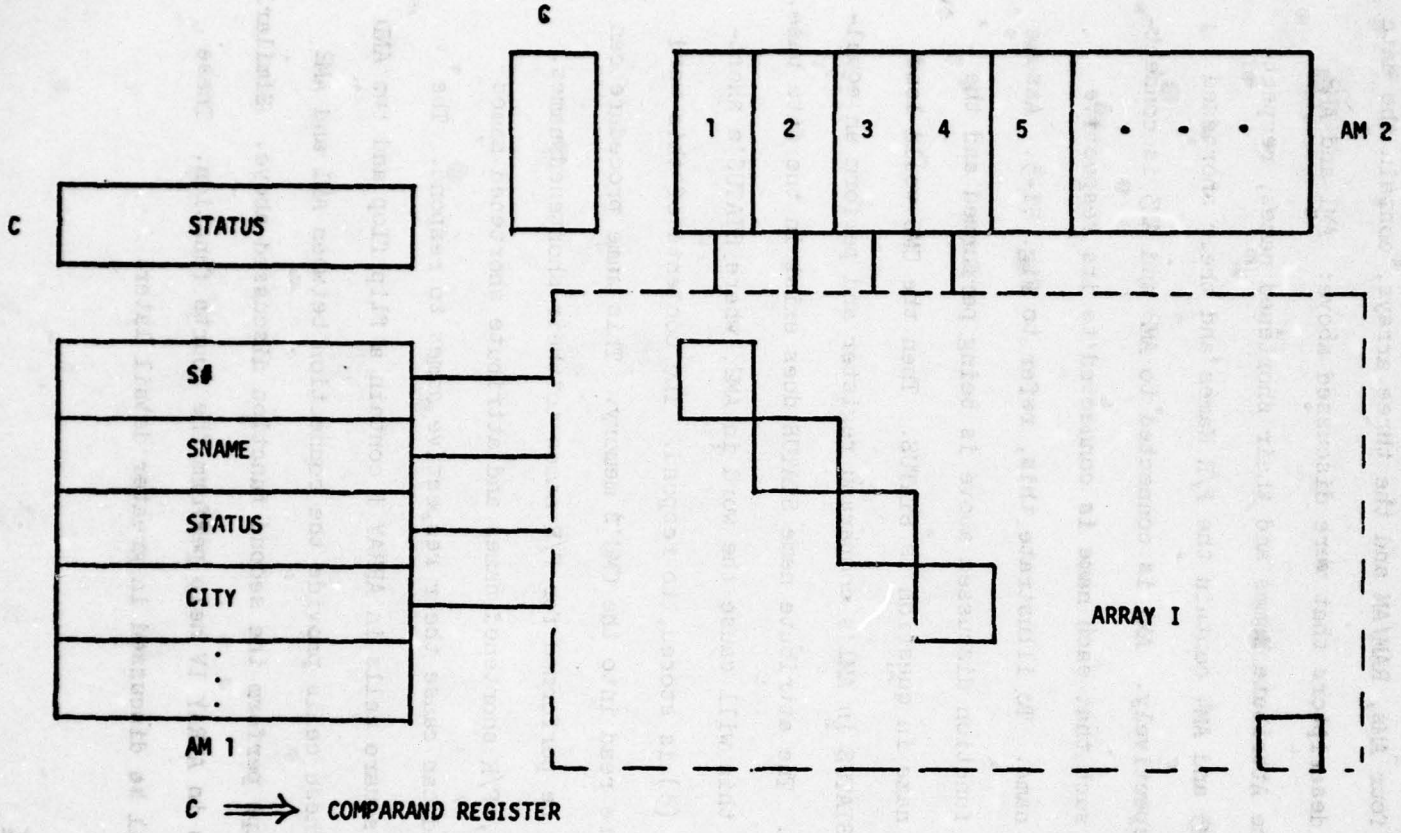


Figure VI-3. The Interaction Among AM1, AM2, and ARRAY I of the Dictionary/Directory Processor.

each row represents an F/R and each column in the matrix represents an attribute's shortened name in the data base. Let each cell of this matrix take on the values zero or one; where a one in cell i,j indicates that the attribute's shortened name stored in the j-th word in AM2 is associated with the F/R name stored in the i-th word in AM3. A zero indicates that they are not associated. (Note that each row of this matrix is a Data Processing Characteristic Set (DPCS)). For illustrative purposes refer to Fig. VI-4. The third function is performed after the first and second, therefore, the activated words in AM2 contain the attribute's shortened names. These activated words are used to drive ARRAY III. To obtain a response in any row of ARRAY III, the row must have a one in each cell whose column is being driven. In the example above, as shown in Fig. VI-4, columns one through four would be activated. The rows one and two of ARRAY III would respond and activate rows one and two of AM3. The CMU could then read out the contents of words one and two of AM3 which contain the F/R names in which the attribute's shortened names (1, 2, 3, and 4) are associated. The reverse process is also possible, i.e. given an F/R name the attribute's shortened names can be obtained.

ARRAY II - RAM/AM

The fifth function which provides the descriptors for each attribute name is accomplished through ARRAY II. Not only is AM1 connected to AM2, word by word, but each word in AM1 and AM2 is

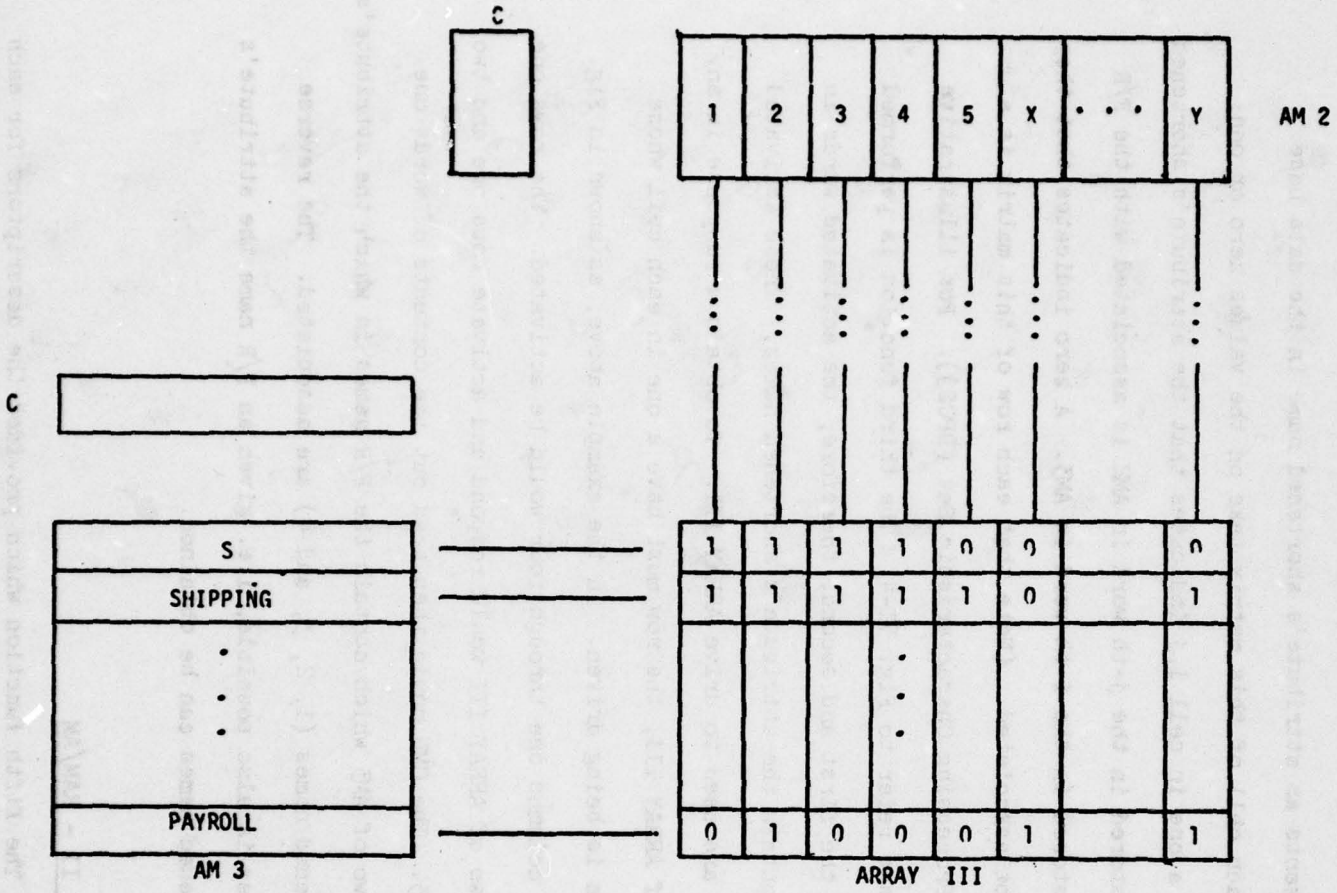


Figure VI-4. Interaction Among AM2, AM3, and ARRAY III of the Dictionary/Directory Processor.

connected to its respective row of cells in ARRAY II. ARRAY II(a) is similar to ARRAY III in that it is a matrix consisting of cells, where each row represents an attribute's name and its shortened name. Each column of ARRAY II(a) represents one of the possible values of each attribute descriptor. These cells consist of one flip flop and one AND gate. They provide the connection between an attribute name (AM1 and AM2) and its descriptors (RAM/AM). To illustrate how function five can be performed, a discussion of the arrangement of the descriptor's values in the RAM/AM is given below.

The RAM/AM contains the attribute's descriptors in a specific order. Assume that the first group of words in the RAM/AM contain the different sizes an attribute's value may have. The second group of words contains the different representations. The third group is the synonym functions, the fourth and fifth are the uniqueness descriptors, the sixth is the password descriptors and the seventh, the provileges descriptor. Within these defined groups, an attribute cannot have more than one value per group. For each value that applies to an attribute, the proper cell in ARRAY II is activated. How these are set shall be discussed later. For now, assume a 1 in cell i,j of ARRAY II, shown in Fig. VI-5, implies that the value in word j of the RAM/AM describes the attribute name in word i of AM1. A zero in i,j implies that the value in word j of the RAM/AM does not apply to the attribute name in word i of AM1. Assume the fifth function is performed after the fourth function; then a row in ARRAY II(a) would be

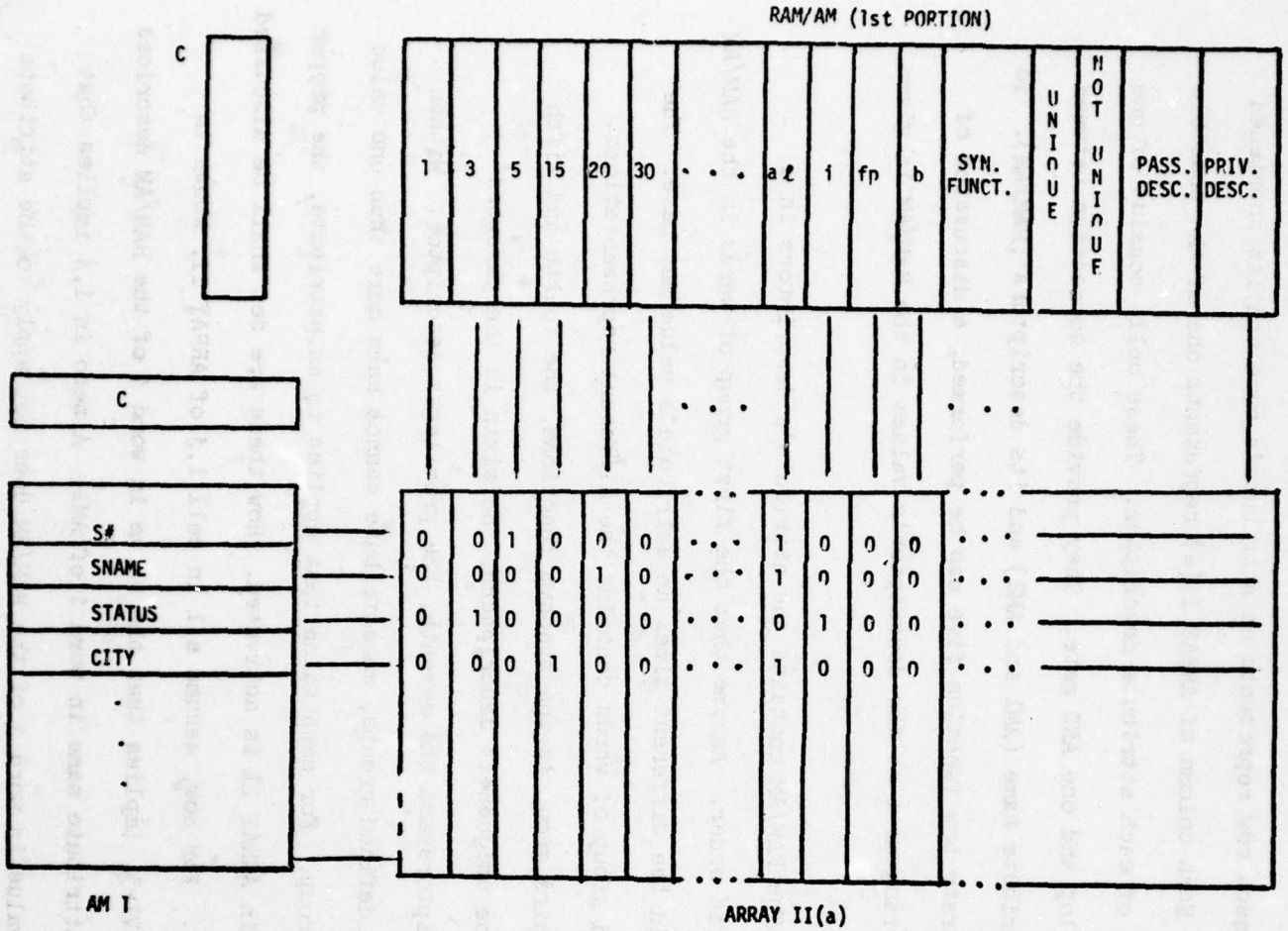


Figure VI-5. Interaction Among AM1, RAM/AM, and ARRAY II of the Dictionary/Directory Processor.

activated. The cells having ones in this row will activate their corresponding words in the RAM/AM. Once this is performed, the CMU can read the contents of these words, in order, into its memory. From Fig. VI-5, the attribute's descriptors would be:

From AM1	From RAM/AM	
S#	5	al
SNAME	20	al
STATUS	3	i
CITY	15	al

AM3, AM4 and RAM/AM

The sixth function which provides the primary key and pointer descriptor for each F/R is accomplished through AM3 and AM4 connected directly to the RAM/AM. Each word in AM3 and AM4 is connected to two consecutive words in the RAM/AM. Since function 6 is performed last, each F/R name, or shortened name, is known and either one can cause the driving signal to activate the RAM/AM to obtain their descriptors. To understand how the sixth function is performed the second portion of the RAM/AM must be described.

The words in the RAM/AM following the attribute name descriptors are reserved for the primary key and pointer descriptors for F/R names. For each F/R in the data base there are two words in the RAM/AM. The first contains the name of the F/R's primary key and the second contains the F/R's pointer descriptor. For

illustrative purposes refer to Fig. VI-6. Assume the sixth function is performed after the fourth function, then the proper words in AM3 and AM4 have been activated. These words can then be used to activate their corresponding words in the RAM/AM. For the example above, the sixth function would provide the following after the CMU read the corresponding words from the RAM/AM and AM3:

F/R Name	Primary Key	Pointer
S	S#	XX

DICTIONARY/DIRECTORY PROCESSOR GATE AND FLIP FLOP LEVEL

To describe the DDP's AM's and ARRAY's at the gate and flip flop level, a series of sections of the hardware will be investigated individually. All the structures represent logical relations only. No optimization nor electrical constraints are considered. The only elements used are RESET/SET (RS) flip flops, AND, and OR gates.

GENERAL AM

Consider Fig. VI-7 as a representation of an AM. Each word would contain the bit configuration representing its respective stored data, e.g. if its AM1 then the stored data are attribute names. The lines for setting each flip flop are not shown but would be connected to the set and reset ports of each flip flop. This figure is a model of all the AMs shown in Fig. VI-2, ARRAY III, and the RAM/AM if it is implemented as an AM. It can be used

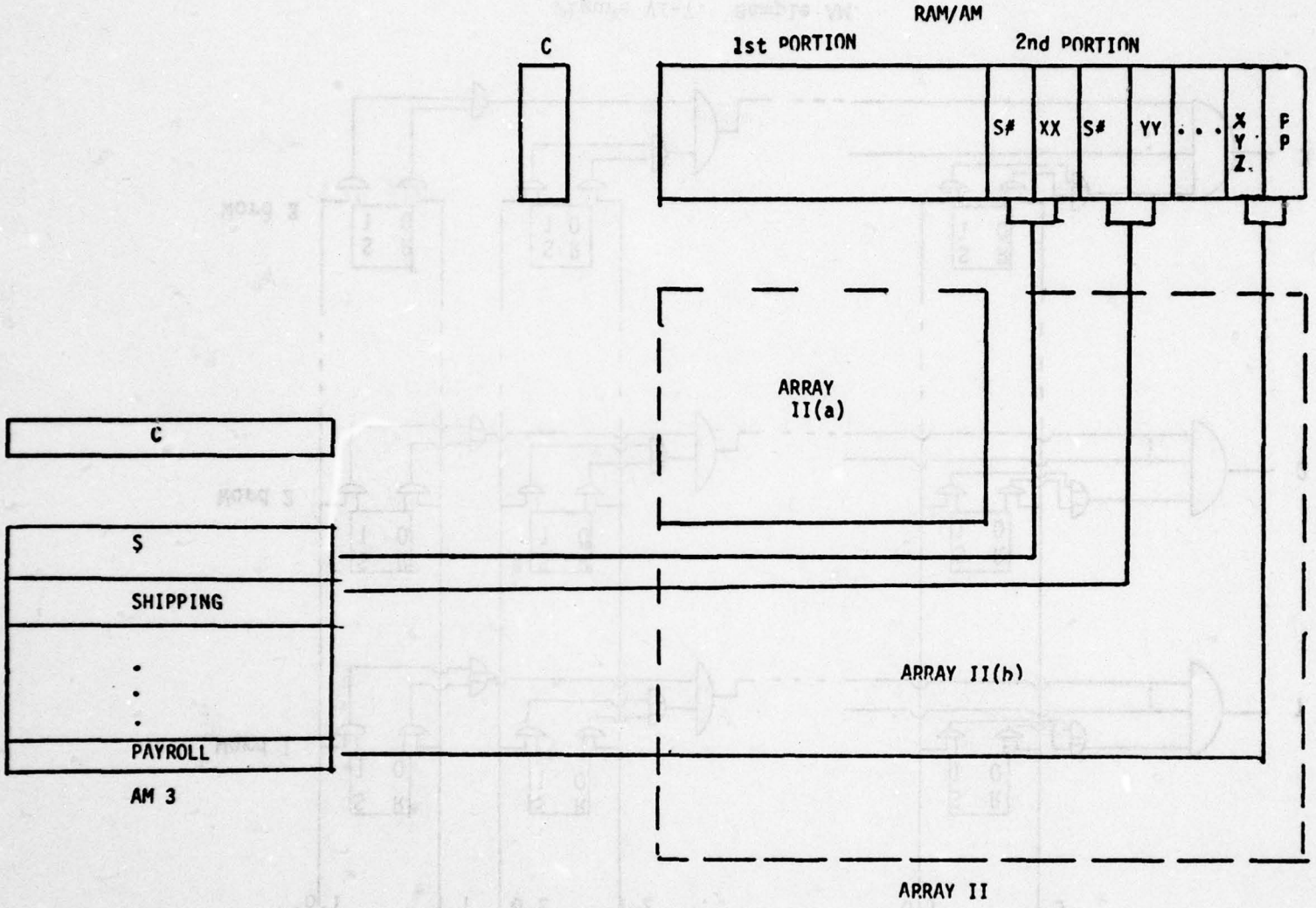


Figure VI-6. Interaction Between AM3 and the RAM/AM of the Dictionary/Directory Processor.

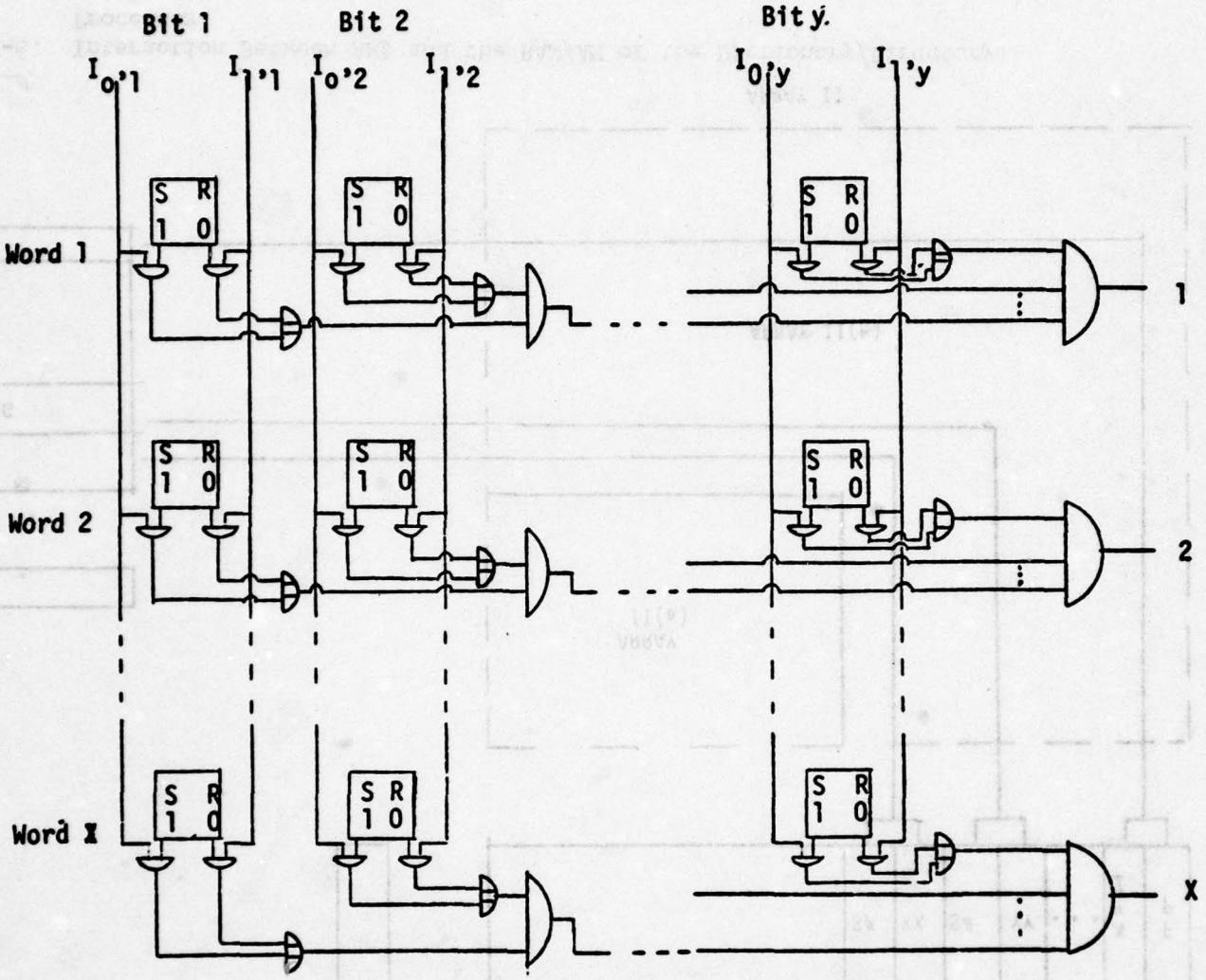


Figure VI-7. Sample AM.

to discuss how an equal to search is accomplished in an AM. Assume that the words in the AM are loaded with their proper data and the equal to search word is stored in the comparand register. For each bit (i) in the comparand register, one of its interrogation lines ($I_{o,i}$ or $I_{1,i}$) is energized by the CMU. If bit i in the comparand register is equal to zero then interrogation line $I_{o,i}$ is energized and if bit i in the comparand register is equal to one, then interrogation line $I_{1,i}$ is energized. If any of the words in the AM are equal to the contents of the comparand register, then the AND gate for the equivalent word will be activated. In addition, if an equal to search was desired on only a portion of the contents of an AM word, then those bits that were not required would have both their interrogation lines $I_{o,i}$ and $I_{1,i}$ energized.

Response Register

The output of these AMs, through their word AND gates, energize a response register. A response register with the additional capability of finding the first response is sometimes called a ladder circuit and can be modeled as shown in Fig. VI-8. To find the first response, assume the flip flops in Fig. VI-8 are initially set to zero by energizing line R. If, for instance, an equal to search is being performed and word 2 in the AM is equal to the word in the word in the data comparand register, then line 2 (from word 2's AND gate) will activate its respective flip flop in the response register. To find the first responsive word after an equal to search in the AM, the

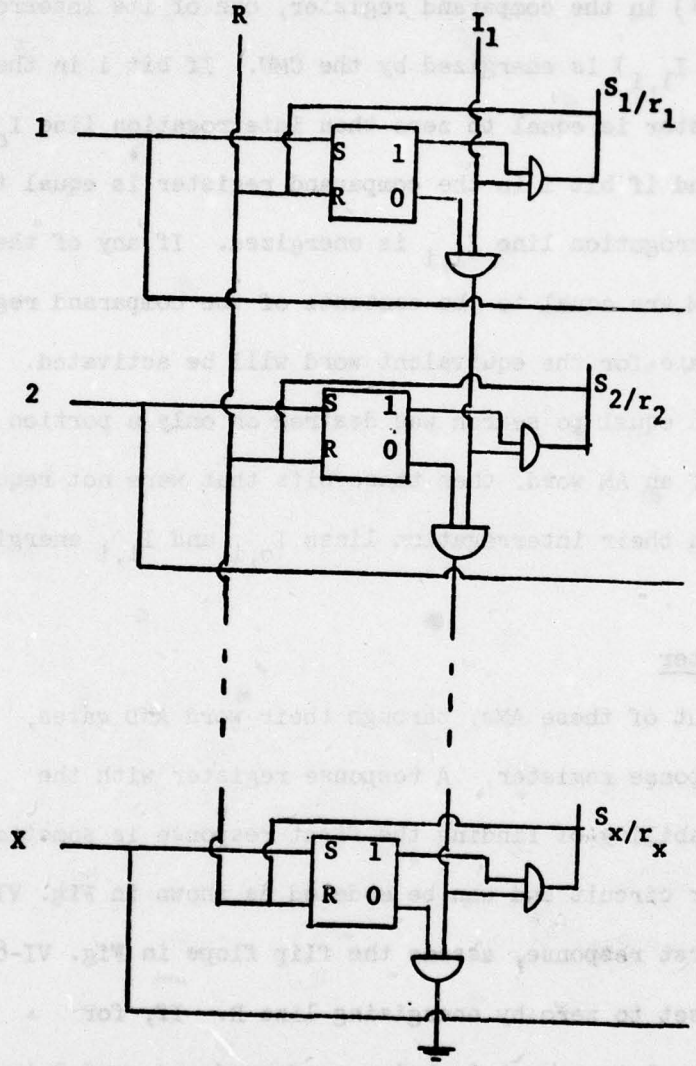


Figure VI-8. Sample Response Register (Ladder Circuit).

interrogation line I_1 is energized and the first word (2) will be sensed by line S_2. To find the next responsive word, after the present word has been processed, the reset line (r_2) would be energized and again I_1 would be energized. It is assumed that the energizing of r_i only resets flip flop i. This procedure would continue until all responsive words were determined.

Function I

Given the above information, it is now possible to add more detail to the six functions discussed above. Consider Figs. VI-9 and VI-10. Figure VI-9 provides a detailed view of ARRAY I shown in Fig. VI-3. Figure VI-10 represents a cell in ARRAY I. The connection of AM1 and AM2 is shown by the lines 1, 1', 2, 2', etc. To activate this connection the CMU must energize interrogation line I_2. This is performed after an equal to search is performed in AM1 as discussed above. The energizing of I_2 causes the respective flip flops in the response register of AM2 to respond therefore designating the correct words to be read by the CMU from AM2. This same procedure is applicable for AM3 to AM4 except interrogation line I_3 is energized instead of I_2.

Functions II and IV

The second and fourth functions are very similar in concept and can be performed structurally using the same procedure and hardware. But first, a synonym attribute must be defined. Two or more attributes which have, at the user's level, the same "value" but have different names are synonym attributes. As stated

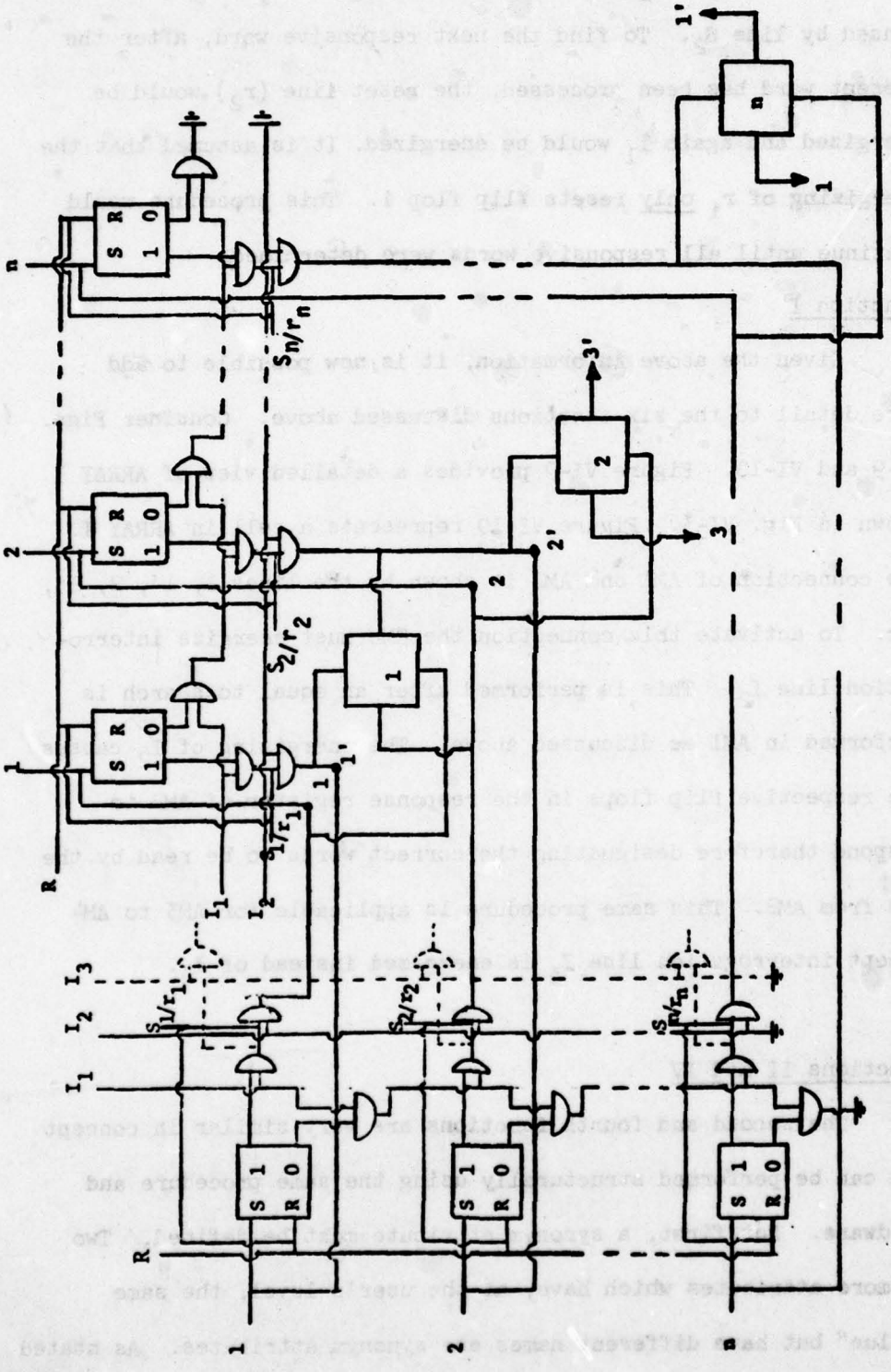


Figure VI-9. Gate Level Interaction Among AM1, AM2 and ARRAY I of the Dictionary/ Directory Processor.

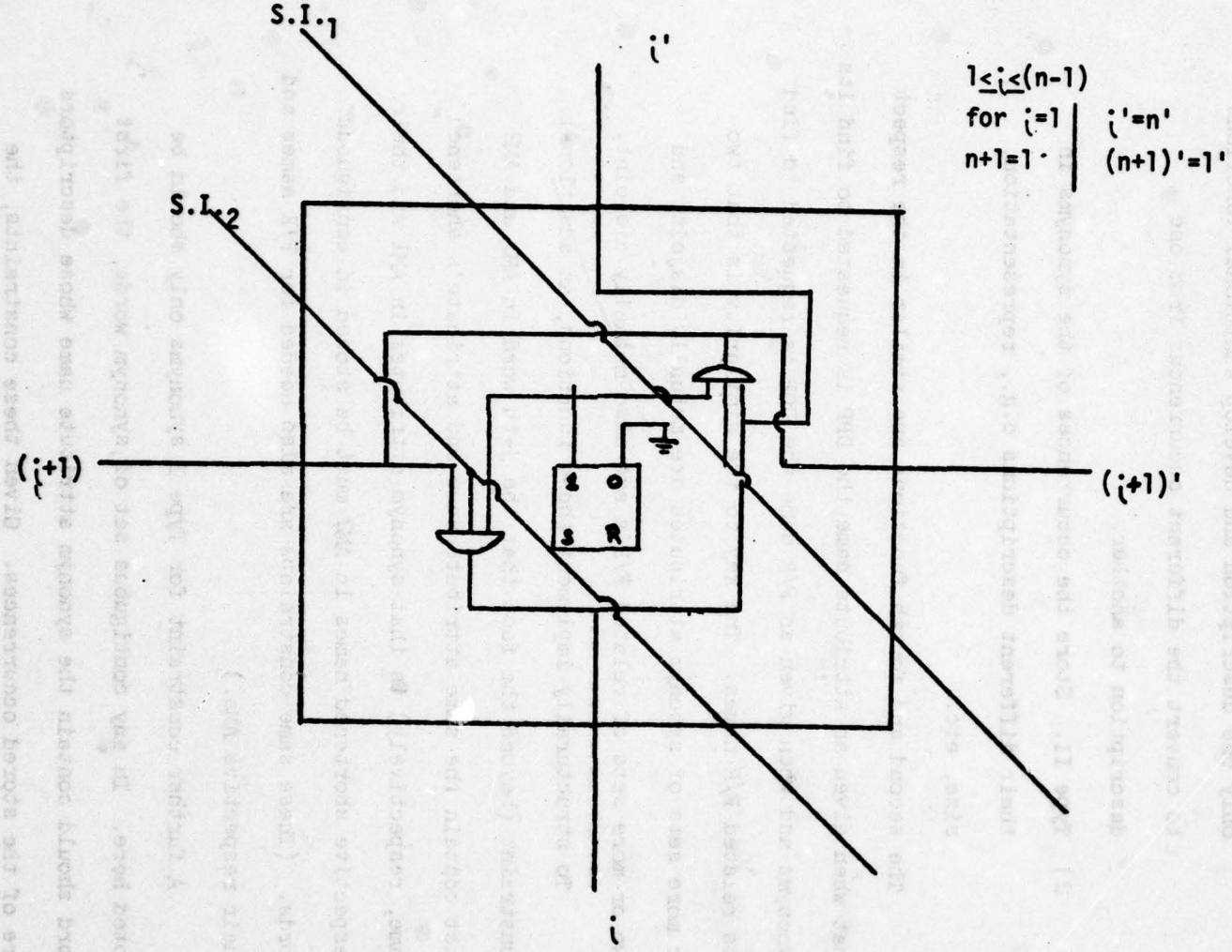


Figure VI-10. Sample Cell of ARRAY I of the Dictionary/Directory Processor.

previously, there are two major ways in which to handle synonyms:

1) Type I. Store the occurrences of the synonyms in only one description and develop additional functions to convert the different occurrences from one description to another.

2) Type II. Store the occurrences of the synonyms in their different descriptions e.g., representation, size, etc.

The second and fourth functions are similar in the respect that when given an attribute name the DDP is requested to find its synonyms and when given an F/R name the DDP is requested to find its related F/R names. The key to the similarity is that two or more sets of synonym attributes are mutually disjoint and two or more sets of related F/R's are also mutually disjoint.

To structurally implement these functions, an additional constraint (beyond the fact that the j-th word in AM1 and AM2 must contain the same attribute name and attribute's shortened name, respectively) is that synonym attributes in AM1 and their respective shortened names in AM2 must be stored in contiguous words. (These same constraints are also needed for F/R names and their respective AMs.)

A further constraint for Type I synonyms only should be noted here. In any contiguous set of synonym words, the first word should contain the synonym attribute name whose descriptors are of the stored occurrences. Given these constraints, the second function can be performed after the first function has

been performed. The procedure can be illustrated through an example. Suppose in Figs. VI-9 and VI-10 the first two words in AM1 contain synonyms, function one has been performed on the first synonym and in Fig. VI-10 i is equal to 1. Then to perform function 2 the CMU will energize SI1 and SI2. If the flip flop in cell 1 is set, this will cause the response registers in the second word of AM1 and AM2 to be set. Similarly, if the second and n-th cells are not set, then the signals in SI1 and SI2 will terminate and the only response registers that will be set will be the first two words of AM1 and AM2. The responding words in the AMs can then be read by the CMU to fulfill the second function. The same procedure is valid for performing function four for AM3, AM4, and ARRAY IV.

In Fig. VI-10 the set and reset lines are not shown. They exist and are controlled by the CMU. The setting, or resetting, of these flip flops allows the DDP to respond to changes in the data base.

Function III

To gain more insight into how function III is performed, consider Fig. VI-11 and Fig. VI-4. In Fig. VI-11, AM3 and AM2 are left out but the lines connecting their respective response registers are shown. The incoming lines at the top of Fig. VI-11, when activated properly, represent the state of the response store of AM2. The lines "n ... 2 1" are returning to the set port of the flip flops in AM2's response register. On the side of Fig.

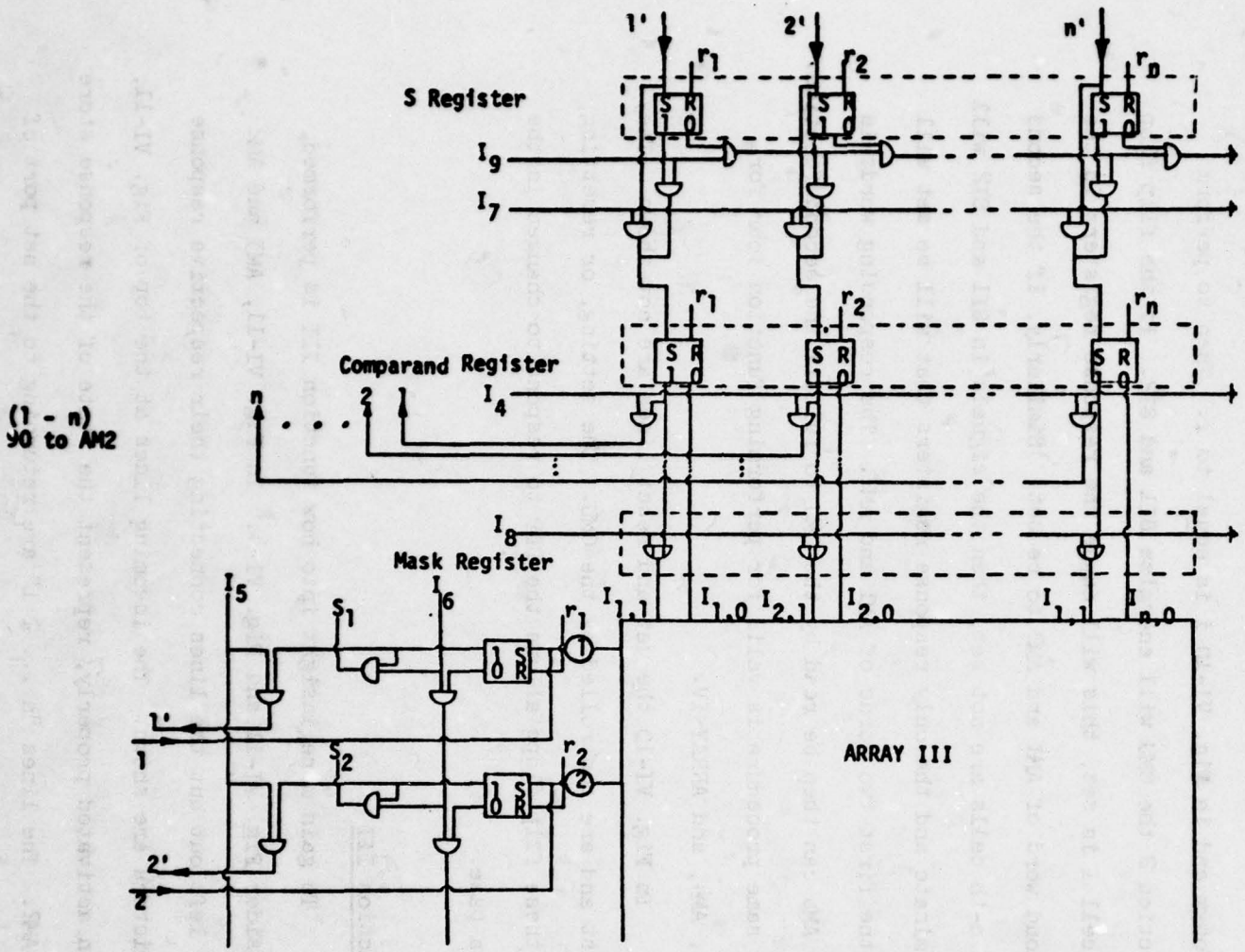


Figure VI-11. Gate Level Interaction Among AM2, AM3 and ARRAY II of the Dictionary/ Directory Processor.

VI-11, the lines 1, 2, 1', and 2' are wired in a similar way with AM3 (See Fig. VI-9).

To perform function three the CMU performs function one, resets the comparand register (shown in Fig. VI-11), and then energizes interrogation lines I_7 and I_8. These operations constitute an equal to search to be performed on the contents of ARRAY III with the responding attribute shortened names in AM2. Energizing I_7 initiates the comparand register and energizing I_8 provides a signal to those bit positions that are not in the equal to search. The set of OR gates associated with I_8 performs the task of "masking" those bit positions not required, therefore the name "mask register." Next, the CMU energizes I_5 which will set the flip flops in AM3's response register (see Fig. VI-9) which denotes the F/R names which have the attribute shortened names in AM2 in their set of attributes. This procedure can also be reversed, i.e. the F/R name is known and it is desired to determine the names of all of its attributes. The task would be done by first performing an equal to search on AM3 with the F/R name in question and then energizing I_2 of AM3 (See Fig. VI-9) in order to cause a flip flop in ARRAY III's response register to be set. The word in ARRAY III, designated by this set flip flop, can be found by energizing I_6 and then loaded into ARRAY III's comparand register. The attribute shortened names are found by the energizing of I_4 which activates lines "n ... 2 1" in ARRAY I which in turn will set flip flops in the response register of AM2.

Function three is not completed as yet. Suppose the data base is implemented so that synonyms are of Type I. Then only one of the attributes, the first one in its set, will have its occurrences stored. Therefore, when the other synonym attribute's occurrences are required, the DBMS will have to access those F/R's where the stored occurrences of the first attribute in the synonym set are located. To determine these F/R names, the DDP would first perform functions one and two. This would cause the S Register shown in Fig. VI-11 to be set by all the synonym attribute shortened names. The CMU would then reset the comparand register and energize interrogation line I_9, in turn. This will cause the first set flip flop in the S Register to set the comparand register. Energizing I_8 and proceeding as above will provide the DDP with the required F/R name(s).

Function V

To understand function V in more detail, refer to Figs. VI-5, VI-12, and VI-13. The lines labeled (1,1'), (2,2'), and (n,n') shown in Fig. VI-12 are connected to AM1 and AM2 through ARRAY I. The lines labeled ℓ_1, ℓ_2,..., ℓ_D are connected to the set ports of the flip flops in the response register for the words in the first portion of the RAM/AM. The cells in the ARRAY II(a) are shown in Fig. VI-13. The setting of these cells by the CMU is similar to having a 1 in ARRAY II shown in Fig. VI-5 discussed previously. To find the descriptors of an attribute, assume that the ladder circuit has been reset in Fig. VI-12 and

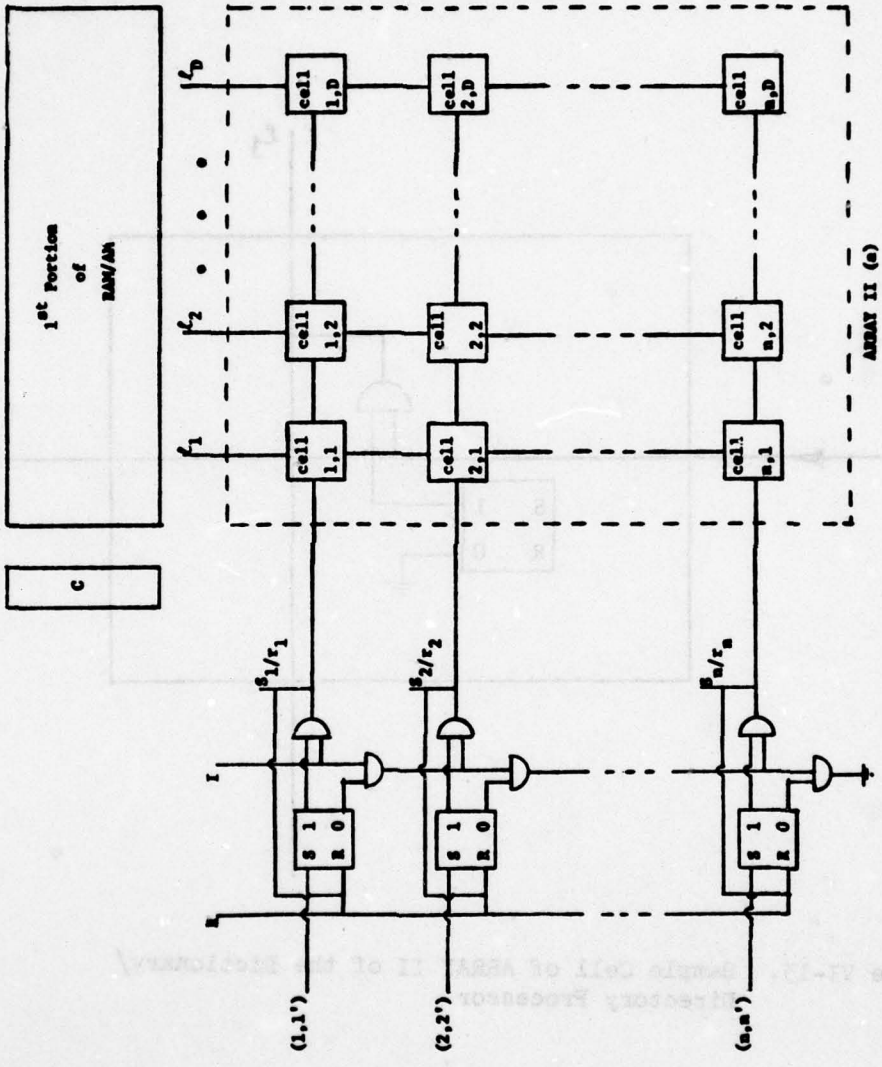


Figure VI-12. Gate Level Interaction Among AML, RAM/AM and ARRAY II(a) of the Dictionary/Directory Processor.

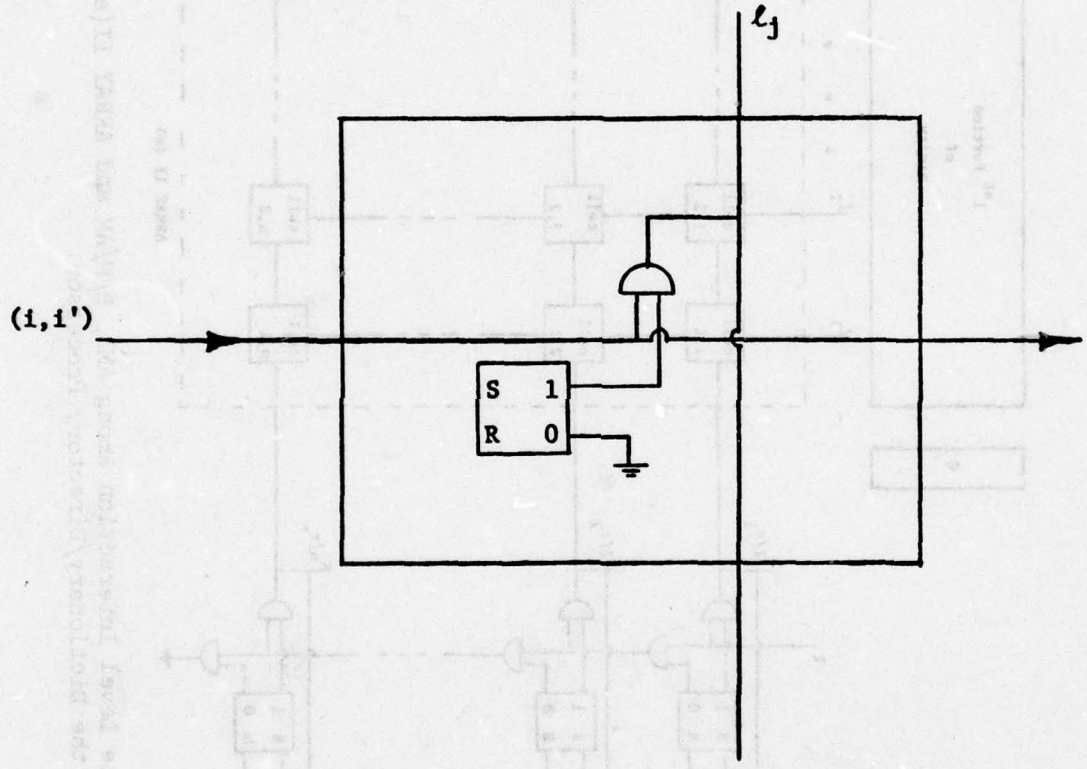


Figure VI-13. Sample Cell of ARRAY II of the Dictionary/ Directory Processor.

and function I has been performed. This will cause one of the flip flops, say flip flop 2, in the ladder circuit to be set. If the CMU then energizes line I,. this will cause the second line to energize the 2,i cells for i = 1, ..., D. The cells in the second row whose flip flops are set will cause their respective lines (ℓ_j) to be energized and therefore set their respective flip flops in the RAM/AM's response register. The CMU can then retrieve the descriptors from the RAM/AM by interrogating its response register. These descriptors will be for that attribute whose name is stored in the second word of AM1.

Function VI

The detail level description for function VI can be described through Fig. VI-6 and Fig. VI-14. The lines (1,1'), (2,2'), ..., (m,m') shown in Fig. VI-14 are connected to ARRAY IV which are connected to AM3 and AM4. Lines labeled ℓ_1, ℓ_2, ℓ_3, ℓ_4, ..., ℓ_{2m-1}, ℓ_{2m} are connected to the set ports of the flip flops in the response register for the words in the second portion of the RAM/AM. To find the descriptors of an F/R, assume that the ladder circuit in Fig. VI-14 has been reset and that the other functions have been performed. Then the F/R name(s) in question will have set one or more of the flip flops in the ladder circuit. Assume that there was one F/R name, e.g. the name stored in the first word of AM3. The CMU would energize interrogation line I to obtain the F/R name's descriptors. The energizing of line I causes lines ℓ_1 and ℓ_2 to set the flip flops in the response register of the

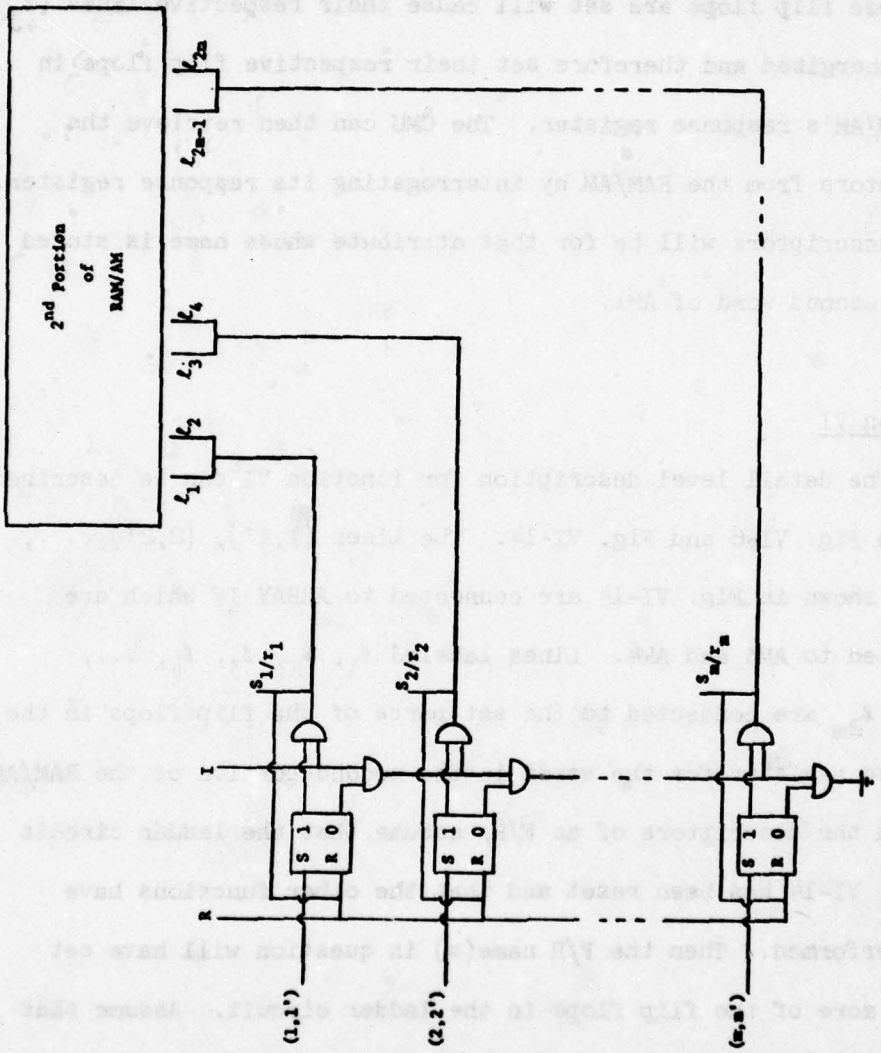


Figure VI-14. Gate Level Interaction Between AMG and the RAM/AM of the Dictionary/Directory Processor.

second portion of the RAM/AM. The CMU would then read into its memory the contents of those words in the RAM/AM and AM3 whose response register is set. This would provide the same data as shown above in discussing Fig. VI-6.

SUMMARY

The contents of this chapter has provided a description of a dictionary/directory processor (DDP). The description provided was in two levels of detail. One was generic with illustrations from the example shown in Fig. V-1. The other description was detailed and emphasized the gate and flip flop level. Both levels were discussed in relation to six defined functions that are performed by DBMSs.

The next chapter contains a presentation of how these functions can be utilized by describing four major jobs that can be submitted to a DBMS. These jobs will then be used to develop a method for evaluating the DDP.

Chapter VII

DICTIONARY/DIRECTORY PROCESSOR EVALUATION

INTRODUCTION

It is customary to develop a method for evaluating a new approach to performing an existent job. This situation exists with the DDP. The contents of this chapter present a method for evaluating the DDP and the results obtained from its evaluation.

In reality it is very difficult to be able to determine the DDP's value; but a comparison to a total software implementation on a sequential computer does provide an indication of its value. The criteria chosen for this evaluation is the comparison of time for the DDP and a sequential computer to execute the same jobs. A total software implementation on a sequential computer was chosen because this is the current method in which a DBMS performs its functions on its data dictionary and data directory.

The sequential computer chosen for this evaluation of the DDP is a fictitious machine called the MIX computer. This machine has been developed and described by Knuth [24]. MIX was chosen because it is not an existing machine but it is like so many existing machines. Knuth states that [24], "MIX is very much like nearly every computer now in existence, except that it is, perhaps, nicer. The language of MIX has been designed to be powerful enough to allow brief programs to be written for most algorithms, yet simple enough so that its operations are easily learned." It

is this machine and its assembly language (MIXAL) that were used to derive the timings for comparison with the DDP.

The timings for the DDP and its interfacing computer were obtained by using times to perform "micro-functions" on an existing AP. The existing AP that was chosen was the STARAN [18]. The major reasons for utilizing an existing machine for these timings was to:

1) show the DDP's feasibility; and
2) observe "realistic" timings for a non-existent piece of hardware.

The choice of this machine, however, not only affects the timings for the DDP's functions, but also its word or field size, data transmission time, and degree of parallelism. It should also be noted that the DDP does not require, nor does it utilize, all of the STARAN's capabilities, e.g. its arithmetic capability.

The timings for the DDP are also influenced by assuming that the RAM/AM is an AM as shown in the many figures in Chapter VI. The timings, however, will not vary much if the RAM/AM is modeled as a RAM; because in the timing equations no searches are performed on the RAM/AM. Data are only retrieved from the RAM/AM. The difference of whether the RAM/AM is a RAM or an AM may be important when considering cost, maintenance, etc. If it were implemented as a RAM then the first portion of the RAM/AM and the second portion could be separated as two RAMs under the control of the central memory unit (CMU).

The vehicle utilized for comparing the two systems consists of four job types. These job types are composed of various subfunctions. Timing equations are developed for the subfunctions and added together to obtain timing equations for the jobs. With these timing equations the two systems can then be compared.

The contents of this chapter is concerned with the following:

1) the development of subfunctions and their timing equations;
2) the application of these subfunctions and timing equations to the development of overall timing equations for the four job types utilizing the DDP;
3) the search technique and the data structures chosen for a sequential computer's data dictionary and partial directory;
4) the development of the timing equations, using MIXAL, to perform each of the four job types on the sequential computer; and
5) the comparison of the results obtained for the DDP and the sequential computer.

DICTIONARY/DIRECTORY PROCESSOR MACRO TIMING EQUATIONS

The timing equations to be developed for the jobs will be for the time increment beginning when the sequential computer's CPU notifies the DDP control of a job to be executed until the

DDP supplies the TUMA with its results. The same timing increment shall be used for the MIX computer. In this case the total data dictionary and data directory are assumed to be stored in main memory. Thus, this will not require any accessing of peripheral devices to obtain data. However, it will require time to move data within main memory from the data dictionary and data directory to the user's memory area (UMA).

The parameters used to describe the above-mentioned jobs, the data dictionary and the data directory are as follows:

1) P, the number of computer words per F/R name;
2) P_1, the number of computer words per attribute name;
3) n_1, the number of F/Rs related to a given F/R and supported by the DBMS;
4) n_2, the number of attributes in a given F/R;
5) n_3, the number of associated F/Rs of a given attribute;
6) n_4, the number of associated F/Rs of a given number (n_5) of attributes;
7) n_5, the number of attributes specified in Job 2;
8) n_6, the number of synonyms of a given attribute; and
9) TA, the total number of attributes in the data base.

These parameters will be varied in the evaluation presented at the end of this chapter. But, more importantly, they will be used in the body of this chapter to develop the timing equations for the evaluation.

In the description of the DDP in Chapter VI, six functions were defined and utilized. These functions, need to be analyzed and explained at a more detailed level so that timing equations can be developed. This level can be thought of as a "micro-function" level. Consider the following instructions as micro-functions and their execution timings obtained from [18] and a STARAN unpublished source:

MICRO-FUNCTIONS	TIME IN μ-SECONDS
ENR (energize a line)	$t_{ENR} = .13$
FRF (find and reset first response)	$t_{FRF} = .18$
LCA (load comparand from an AM)	$t_{LCA} = .64$
CLR (clear register (Set = 0))	$t_{CLR} = .13$
SCM (store comparand in control memory unit (CMU))	$t_{SCM} = .7$
TRN (transfer data to and from the sequential computer and the CMU)	$t_{TRN} = .3$/word (word length $\leq$ 32 bits)
FRA (find first and reset all other responders)	$t_{FRA} = .8$
LCM (load comparand from the CMU)	$t_{LCM} = .7$
XXY (boolean AND operation on response registers X and Y with its result stored in X)	$t_{XXY} = .17$

MICRO-FUNCTIONS	TIME IN μ-SECONDS
YXY (boolean AND operation on response registers X and Y with its result stored in Y)	$t_{YXY} = .17$
EQC (equal to search on AM with the comparand's contents)	$t_{EQC} = (.18)(\text{bits/word}) + (.8)$ $= (.18)(31) + (.8) = 6.38$
SET (set register = 1)	$t_{SET} = .13$

These micro-functions can be put together to form "macro-functions."

Before these macro-functions are developed, it should be noted that the AMs described earlier have only one response register. This register will be referred to as Y. There will also exist another register, X (see STARAN [18]), to be used in equal to searches. The other note is that the MIX computer is 31 bits long and the maximum effective STARAN field length is 32 bits. To maintain compatibility between the two for the following examples, the length of a computer word/field will be 31 bits.

The rest of this section of the chapter is partitioned into six parts. Each provides a development of timing equations for one or more macro-functions. These macro-functions are composed of a series of the above micro-functions performed in an exact sequence. The time required for each micro-function in the sequence is summed to form a timing equation. These macro-functions are equal to or are a part of one of the six functions mentioned previously. The development described in the next section of this chapter utilizes these timing equations to form the test bed for the DDP's evaluation.

FUNCTION I

The first function determines if, in fact, the F/R names and attribute names exist in the data base. If they do, the DDP provides their names and their shortened names to the DBMS.

Single Equal to Search (SETS)

To determine if an F/R name or attribute name exists, an equal to search (ETS) must be performed on the proper AM after the name in question is sent to the DDP. The SETS macro-function and its performance time for one computer word is:

	INSTRUCTION	TIME	EXPLANATION
(01)	TRN	t_{TRN}	Transfer F/R name to the DDP's CMU
(02)	LCM	t_{LCM}	Loads comparand register from the CMU with the value to be searched
(03)	CLR Y	t_{CLR}	Sets response register Y to zero
(04)	EQC	t_{EQC}	Performs an equality search on all words in the AM and sets response register Y.

The total time to perform SETS is

$$T_{SETS} = t_{TRN} + t_{LCM} + t_{CLR} + t_{EQC}.$$

Multiple Word ETS (METS)

If the variable to be searched is greater than one computer word and equal to or less than eight computer words, the ETS macro-function, METS, must be performed. The micro-functions for METS are:

	INSTRUCTION	TIME	EXPLANATION
(01)	TRN	$(w)^{*}t_{TRN}$	Transfer F/R name to the DDP's CMU
(02)	SET X	t_{SET}	Sets response register X to 1
(03)	CLR Y	$(w)\ t_{CLR}$	Sets the Y register equal to zero
(04)	LCM	$(w)\ t_{LCM}$	Load the comparand register with one word from the CMU
(05)	EQC	$(w)\ t_{EQC}$	Performs an equality search on all words in the AM and sets response register Y
(06)	XXY	$(w)\ t_{XXY}$	"ANDS" registers X and Y and stores result in X

The total time to perform METS is

$$T_{METS} = (w)(t_{TRN} + t_{LCM} + t_{EQC}) + (w)(t_{CLR} + t_{XXY}) + t_{SET},$$

where w is the number of words to be searched. (The size of the variable to be searched was limited to eight computer words because a STARAN AM is 256 bits wide.)

Connect (CONN)

ETS is the macro-function that can provide the first part of the first function. To find an attribute's shortened name, given its name or vice versa, requires the energizing of L_2 of AM1 or AM2, respectively. Similarly, to find an F/R's shortened name, given its name, or vice versa, requires the energizing of

*w is the number of computer words to be searched (i.e. $1 < w \leq 8$). A w in parentheses indicates that that micro-function must be performed for each computer word.

I_3 of AM3 or I_2 of AM4, respectively. The CONN macro-function and its performance time for these four possible connections is:

	INSTRUCTION	TIME	EXPLANATION
(01)	ENR	t_{ENR}	energize a line

The total time to perform CONN is $T_{CONN} = t_{ENR}$

Single Retrieve (SRET)

The final part of the first function must provide to the TUMA the contents of a word(s) in an AM. This can be achieved by the retrieve macro-functions. The SRET macro-function and its performance time for one AM computer word is:

	INSTRUCTION	TIME	EXPLANATION
(01)	FRF	t_{FRF}	Find and reset first response in the AM
(02)	LCA	(w) t_{LCA}	Load the comparand with w words from the AM
(03)	SCM	(w) t_{SCM}	Store in the CMU the w words that are in the comparand
(04)	TRN	(w) t_{TRN}	Transfer the w words in the CMU to the sequential computer

The total time to perform SRET is

$$T_{SRET} = t_{FRF} + w[t_{LCA} + t_{SCM} + t_{TRN}].$$

Multiple Retrieve (MRET)

If there is more than one AM word (n) to be retrieved, then a multiple retrieve macro-function, MRET is desired. The MRET

macro-function and its performance time for n AM words of w computer words is:

	INSTRUCTIONS	TIME	EXPLANATION
(01)	FRF	$(n+1)\ t_{FRF}$	Find and reset first response in the AM (this is done n+1 times)
(02)	LCA	$(w)(n)\ t_{LCA}$	Load the comparand n times with w words from the AM
(03)	SCM	$(w)(n)\ t_{SCM}$	Store in the CMU the w words that are in the comparand (this is done n times)
(04)	TRN	$(w)(n)\ t_{TRN}$	Transfer the w words in the CMU to the sequential computer (this is done n times)

The total time to perform MRET is

$$T_{MRET} = (w)(n)[t_{LCA} + t_{SCM} + t_{TRN}] + (n+1)t_{FRF},$$

where the extra FRF instruction is necessary to determine when n retrieved words are completed.

Comparand Retrieve (CRET)

Finally, if the word to be retrieved is in the comparand register, then the macro-function, CRET is desired. The CRET macro-function and its performance time for w computer words is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the Y register to zero
(02)	SCM	(w) t_{SCM}	Store in the CMU the w words that are in the comparand
(03)	TRN	(w) t_{TRN}	Transfer the w words in the CMU to the sequential computer

The total time to perform CRET is

$$T_{CRET} = w[t_{SCM} + t_{TRN}] + t_{CLR}.$$

FUNCTION II

The second function determines if an attribute(s) has any synonyms. If it does, then their attribute names must be determined depending on whether the DBMS supports Type I or Type II synonyms and the type of command making the request.

Synonyms (SYNO)

To determine if an attribute has any synonyms, a response in either AM1 or AM2 must exist. Then the SYNO macro-function will find the respondent attribute's synonyms. The SYNO macro-function and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	ENR	t_{ENR}	Energize I_2 of either AM1 or AM2
(02)	ENR	t_{ENR}	Energize SI_1 and SI_2

The total time to perform SYNO is $T_{SYNO} = (2)t_{ENR}$.

FUNCTION III

The third function determines if the attributes associated with their respective F/R name in the user's request are correct.

F/R to Attribute (FRTA)

FRTA is a macro-function that provides those attribute shortened names that are associated with a given F/R name already found in AM3. The FRTA macro-function and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	ENR	t_{ENR}	Energize interrogation line I_2 of AM3
(02)	FRF	t_{FRF}	Find and reset the first response register of ARRAY III
(03)	LCA	$(\lceil \frac{TA}{31} \rceil)\ t_{LCA}$	Load the comparand register of ARRAY III from its AM with the contents of the previous respondent word
(04)	ENR	t_{ENR}	Energize interrogation line I_4 of ARRAY III

The total time to perform FRTA is

$$T_{FRTA} = (2)t_{ENR} + t_{FRF} + (\lceil \frac{TA}{31} \rceil)\ t_{LCA},$$

where TA = total number of attributes in the data base and $\lceil x \rceil$ is the least integer $\geq x$.

Attribute to F/R (ATFR)

The ATFR macro-function provides the F/R name, given one or more attribute shortened names. The ATFR macro-function and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	ENR	t_{ENR}	Energize interrogation line I_2 of AM2
(02)	CLR C	t_{CLR}	Set the comparand register of ARRAY III to zero
(03)	ENR	t_{ENR}	Energize interrogation line I_7 of ARRAY III
(04)	ENR	t_{ENR}	Energize interrogation line I_8 to ARRAY III
(05)	SET X	t_{SET}	Set the X register of ARRAY III to one
(06)	EQC	$(\lceil \frac{TA}{31} \rceil)\ t_{EQC}$	Perform an equal to search on ARRAY III with its comparand's contents
(07)	XXY	$(\lceil \frac{TA}{31} \rceil)\ t_{XXY}$	Perform a boolean AND operation on X and Y registers of ARRAY III and store the results in the X register
(08)	CLR Y	$(\lceil \frac{TA}{31} \rceil)\ t_{CLR}$	Set the Y register of ARRAY III to zero
(09)	YXY	t_{YXY}	Perform a boolean AND operation on the X and Y registers of

	INSTRUCTION	TIME	EXPLANATION
			ARRAY III and store the results in the Y register
(10)	ENR	t_{ENR}	Energize interrogation line I_5 of ARRAY III
(11)	CLR Y	t_{CLR}	Set the Y register of ARRAY III to zero

The total time to perform ATFR is

$$T_{ATFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 2)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}.$$

Single Synonym to F/R (SSFR)

The synonym to F/R (STFR) macro-function is similar to ATFR. Given a data base with either Type I or Type II synonyms and the SYNO macro-function has been performed, then the STFR macro-function provides the name of the F/R where the synonym's occurrences are stored. The SSFR macro-function and its performance time for $n_6 = 1$ is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR S	t_{CLR}	Set the S register of ARRAY III to zero
(02)	ENR	t_{ENR}	Energize interrogation line I_2 of AM2
(03)	CLR C	t_{CLR}	Set the comparand register of ARRAY III to zero

	INSTRUCTION	TIME	EXPLANATION
(04)	ENR	t_{ENR}	Energize interrogation line I_9 of ARRAY III
(05)	ENR	t_{ENR}	Energize interrogation line I_8 of ARRAY III
(06)	SET X	t_{SET}	Set the X register of ARRAY III to one
(07)	EQC	$(\lceil \frac{TA}{31} \rceil)\ t_{EQC}$	Perform an equal to search on ARRAY III with its comparand's contents
(08)	XXY	$(\lceil \frac{TA}{31} \rceil)\ t_{XXY}$	Perform a boolean AND operation on the X and Y registers of ARRAY III and store the results in the X register
(09)	CLR Y	$(\lceil \frac{TA}{31} \rceil)\ t_{CLR}$	Set the Y register of ARRAY III to zero
(10)	YXY	t_{YXY}	Perform a boolean AND operation on the X and Y registers of ARRAY III and store the results in the Y register
(11)	ENR	t_{ENR}	Energize interrogation line I_5 of ARRAY III
(12)	CLR Y	t_{CLR}	Set the Y register of ARRAY III to zero

The total time to perform SSFR is

$$T_{SSFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 3)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}.$$

Multiple Synonym to F/R (MSFR)

If there is more than one synonym, i.e. $n_6 \geq 2$, then the STFR macro-function MSFR and its performance time is:

	INSTRUCTION	TIME		EXPLANATION
(01)	CLR S		t_{CLR}	Set the S register of ARRAY III to zero
(02)	ENR		t_{ENR}	Energize interrogation line I_2 of AM2 and/or I_2 of AM1
(03)	CLR C	(n_6)	t_{CLR}	Set the comparand register to ARRAY III to zero
(04)	ENR	(n_6+1)	t_{ENR}	Energize interrogation line I_9 of ARRAY III
(05)	FRF	(n_6+1)	t_{FRF}	Find and reset first response in register S of ARRAY III
(06)	ENR	(n_6)	t_{ENR}	Energize interrogation line I_8 of ARRAY III
(07)	SET X	(n_6)	t_{SET}	Set the X register of ARRAY III to one
(08)	EQC	$(n_6)(\lceil \frac{TA}{31} \rceil)$	t_{EQC}	Perform an equal to search on ARRAY III with its comparand's contents
(09)	XXY	$(n_6)(\lceil \frac{TA}{31} \rceil)$	t_{XXY}	Perform a boolean AND operation on the X and Y registers of ARRAY III and store results in the X register

	INSTRUCTION	TIME	EXPLANATION
(10)	CLR Y	$(n_6)(\lceil \frac{TA}{31} \rceil)\ t_{CLR}$	Set the Y register of ARRAY III to zero
(11)	YXY	$(n_6)(t_{YXY})$	Perform a boolean AND operation on the X and Y registers of ARRAY III and store the results in the Y register
(12)	ENR	$(n_6)\ t_{ENR}$	Energize interrogation line I_5 of ARRAY III
(13)	CLR Y	$(n_6)\ t_{CLR}$	Set the Y register of ARRAY III to zero

The total time to perform MSFR is

$$T_{MSFR} = ((2 + \lceil \frac{TA}{31} \rceil)(n_6) + 1)t_{CLR} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{EQC}$$
$$+ 3(n_6 + \frac{2}{3})t_{ENR} + (n_6)t_{SET} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{XXY}$$
$$+ (n_6)t_{YXY} + (n_6 + 1)t_{FRF},$$

where the extra ENR and FRF instructions are necessary to determine when n_6 synonyms are completed.

FUNCTION IV

If there were more than one F/R involved in a user's request and there was an implied relationship among them, then the fourth function would be to determine if, in fact, the DBMS supported a relationship among these F/Rs.

Relationship (RELA)

The macro-function relationship (RELA) determines whether or not there exists a supported relationship between two or more F/Rs. The RELA macro-function and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	ENR	t_{ENR}	Energize interrogation line I_2 of AM4 or interrogation line I_3 of AM3
(02)	ENR	t_{ENR}	Energize SI_1 and SI_2 of ARRAY IV

The total time to perform RELA is $T_{RELA} = (2)t_{ENR}$.

FUNCTION V

The fifth function provides the rest of the descriptors for each attribute name and synonym name involved in a requesting command.

Single Attribute to Descriptors (SATD)

The macro-function SATD provides the descriptors for one attribute. The macro-function SATD and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero
(02)	ENR	t_{ENR}	Energize the interrogation line I_2 of AM1 or AM2
(03)	ENR	t_{ENR}	Energize the interrogation line I of ARRAY II

The total time to perform SATD is $T_{SATD} = (2)t_{ENR} + t_{CLR}$.

Multiple Attribute to Descriptors (MATD)

The macro-function MATD provides the descriptors for n attributes, where n is greater than one. The macro-function MATD and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero
(02)	ENR	t_{ENR}	Energize the interrogation line I_2 of AM1 or AM2
(03)	ENR	(n+1) t_{ENR}	Energize the interrogation line I of ARRAY II
(04)	FRF	(n+1) t_{FRF}	Find and reset the first respondent of the ladder circuit of ARRAY II

The total time to perform MATD is

$$T_{MATD} = t_{CLR} + (n+2)t_{ENR} + (n+1)t_{FRF},$$

where the extra ENR and FRF instructions are necessary to determine when the n attributes are completed.

Synonym I to Descriptors (SITD)

The macro-function SITD provides the descriptors for the stored attribute of Type I synonyms. The macro-function SITD and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero

	INSTRUCTION	TIME	EXPLANATION
(02)	ENR	t_{ENR}	Energize interrogation line I_2 of AM1 or AM2
(03)	ENR	t_{ENR}	Energize interrogation lines SI_1 and SI_2 of ARRAY I
(04)	ENR	t_{ENR}	Energize interrogation line I of ARRAY II
(05)	FRA	t_{FRA}	Find the first respondent in AM2 or AM1 and reset all others
(06)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero

The total time to perform SITD is $T_{SITD} = (3)t_{ENR} + (2)t_{CLR} + t_{FRA}$.

FUNCTION VI

The sixth function provides the F/R primary key and pointer for each F/R and related F/R involved in a query to the data base.

Single F/R to Descriptors (SFRD)

The macro-function SFRD provides the descriptors for one F/R name. The macro-function SFRD and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero
(02)	ENR	t_{ENR}	Energize interrogation line I_2 of AM4 or interrogation line I_3 of AM3

	INSTRUCTION	TIME	EXPLANATION
(03)	ENR	t_{ENR}	Energize interrogation line I of ARRAY II

The total time to perform SFRD is $T_{SFRD} = (2)t_{ENR} + t_{CLR}$.

Multiple F/R to Descriptors (MFRD)

The macro-function MFRD provides the descriptors for n F/Rs, where n is greater than one. The macro-function MFRD and its performance time is:

	INSTRUCTION	TIME	EXPLANATION
(01)	CLR	t_{CLR}	Set the ladder circuit of ARRAY II to zero
(02)	ENR	t_{ENR}	Energize interrogation line I_2 of AM4 or interrogation line I_3 of AM3
(03)	ENR	(n+1) t_{ENR}	Energize interrogation line I of ARRAY II
(04)	FRF	(n+1) t_{FRF}	Find and reset the first respondent of the ladder circuit of ARRAY II

The total time to perform MFRD is

$$T_{MFRD} = (n+2)t_{ENR} + (n+1)t_{FRF} + t_{CLR},$$

where the extra ENR and FRF instructions are necessary to determine when the n F/Rs are completed.

This completes the development of the timing equations for the specific macro-functions making up the six functions discussed

in the previous chapter. The macro-functions are summarized in Table VII-1. The next portion of this chapter contains the development of the timing equations for the four generic jobs making up the test bed for the DDP's evaluation.

DICTIONARY/DIRECTORY PROCESSOR JOB TIMING EQUATIONS

The following are the four generic jobs to be used in developing timing equations for the DDP. Each job is followed by an example referring to the Suppliers Data Model shown in Fig. V-1.

1) Given the F/R name, the DBMS is required to provide all its occurrences.

Example: To provide all the occurrences of the F/R S and to define it as an F/R W.

GET W(S.S#, S.SNAME, S.STATUS, S.CITY)

RESULT

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

2) Given an F/R name and a number of its attribute names, the DBMS is required to provide a subset of the occurrences of the F/R.

Example: To provide all the occurrences of the F/R S whose occurrence of CITY is equal to LONDON and to define the results as an F/R W.

Table VII-1. Macro-Function Timing Equations for the Dictionary/ Directory Processor.

FUNCTION I

Macro-function: SETS

Explanation: Single Equal to Search

Timing Equation: $T_{SETS} = t_{TRN} + t_{LCM} + t_{CLR} + t_{EQC}$

Macro-function: METS

Explanation: Multiple word ETS

Timing Equation: $T_{METS} = (w)[t_{TRN} + t_{LCM} + t_{EQC}] + (w)[t_{CLR} + t_{XXY}] + t_{SET}$

Macro-function: CONN

Explanation: Connect

Timing Equation: $T_{CONN} = t_{ENR}$

Macro-function: SRET

Explanation: Single Retrieve

Timing Equation: $T_{SRET} = t_{FRF} + (w)[t_{LCA} + t_{SCM} + t_{TRN}]$

Macro-function: MRET

Explanation: Multiple Retrieve

Timing Equation: $T_{MRET} = (w)(n)[t_{LCA} + t_{SCM} + t_{TRN}] + (n+1)t_{FRF}$

Macro-function: CRET

Explanation: Comparand Retrieve

Timing Equation: $T_{CRET} = (w)[t_{SCM} + t_{TRN}] + t_{CLR}$

Table VII-1 (continued)

FUNCTION II

Macro-function: SYNO

Explanation: Synonyms

Timing Equation: $T_{SYNO} = (2)t_{ENR}$

FUNCTION III

Macro-function: FRTA

Explanation: F/R to Attribute

Timing Equation: $T_{FRTA} = (2)t_{ENR} + t_{FRF} + (\lceil \frac{TA}{31} \rceil)t_{LCA}$

Macro-function: ATFR

Explanation: Attribute to F/R

Timing Equation: $T_{ATFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 2)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}$

Macro-function: SSFR

Explanation: Single Synonym to F/R

Timing Equation: $T_{SSFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 3)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}$

Macro-function: MSFR

Explanation: Multiple Synonym to F/R

Timing Equation: $T_{MSFR} = ((2 + \lceil \frac{TA}{31} \rceil)(n_6) + 1)t_{CLR} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{EQC} + 3(n_6 + \frac{2}{3})t_{ENR} + (n_6)t_{SET} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{XXY} + (n_6)t_{YXY} + (n_6 + 1)t_{FRF}$

Table VII-1 (continued)

FUNCTION IV

Macro-function: RELA

Explanation: Relationship

Timing Equation: $T_{RELA} = (2)t_{ENR}$

FUNCTION V

Macro-function: SATD

Explanation: Single Attribute to Descriptors

Timing Equation: $T_{SATD} = (2)t_{ENR} + t_{CLR}$

Macro-function: MATD

Explanation: Multiple Attribute to Descriptors

Timing Equation: $T_{MATD} = t_{CLR} + (n + 2)t_{ENR} + (n + 1)t_{FRF}$

Macro-function: SITD

Explanation: Synonym I to Descriptors

Timing Equation: $T_{SITD} = (3)t_{ENR} + (2)t_{CLR} + t_{FRA}$

FUNCTION VI

Macro-function: SFRD

Explanation: Single F/R to Descriptors

Timing Equation: $T_{SFRD} = (2)t_{ENR} + t_{CLR}$

Macro-function: MFRD

Explanation: Multiple F/R to Descriptors

Timing Equation: $T_{MFRD} = (n + 2)t_{ENR} + (n + 1)t_{FRF} + t_{CLR}$

GET W(S.S#,S.SNAME,S.STATUS,S.CITY):S.CITY = 'LONDON'

RESULT W

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S4	CLARK	20	LONDON

3) Given an attribute's occurrence(s) of Type II synonyms, the DBMS is required to modify (e.g. delete, change value, etc.) the attribute's and its synonym's occurrences in all their associated F/Rs.

Example: To change the occurrence of STATUS to 20 for those occurrences whose S# = S3.

HOLD W(S.S#,S.STATUS): S.S# = 'S3'

W. STATUS = '20'

UPDATE W

RESULT S

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	20	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

If the attribute STATUS had any synonyms associated with other F/Rs, then the DBMS would have to change all their occurrences whose S# equaled S3, or an equivalent unique identifier to the S3 occurrence of F/R S.

4) Given an attribute's occurrence(s) of Type I synonyms, the DBMS is required to modify (e.g. delete, change

value, etc.) the attribute's stored synonym occurrence(s) in its associated F/R.

Example: To change the occurrence of an attribute name STAT to 20.0 in the F/R K for those occurrences whose S# = S3. The attribute STAT has its values stored with its synonym, STATUS, in F/R S.

```
HOLD      W(K.S#,K.STAT): K S# = 'S3'
          W. STATUS = '20.0'
UPDATE    W
```

These three statements would be given by the user. The DBMS would not change the value STAT in F/R K but would internally generate equivalent statements as shown in the example for Job 3 and provide the following:

RESULT	S	S#	SNAME	STATUS	CITY
		S1	SMITH	20	LONDON
		S2	JONES	10	PARIS
		S3	BLAKE	20	PARIS
		S4	CLARK	20	LONDON
		S5	ADAMS	30	ATHENS

JOB 1

The first generic job is for the DBMS to provide all the occurrences of an F/R given only the F/R name. To develop the timing equation for Job 1, consider Fig. VII-1 and Table VII-2. The process for Job 1 is to first find the F/R name and all (n_2) of its attribute names, shortened names and their descriptors.

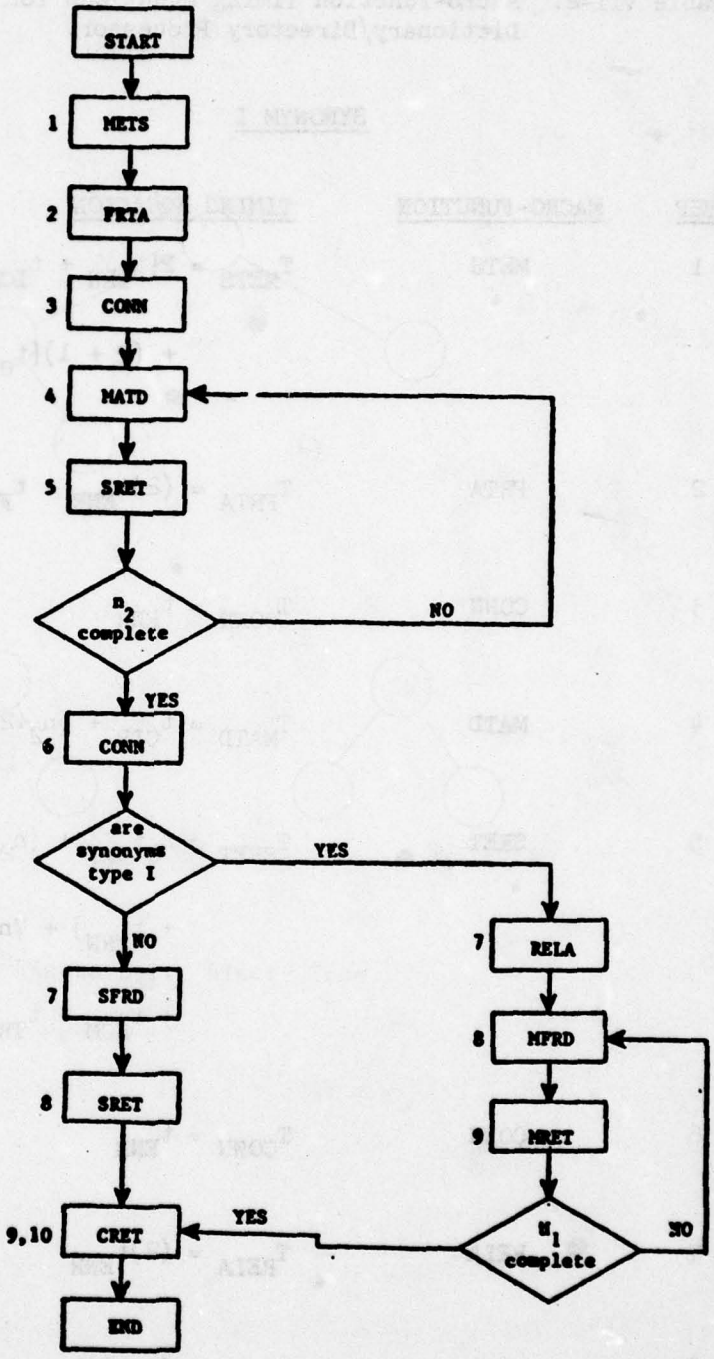


Figure VII-1. Job 1 Flow Diagram for the Dictionary/Directory Processor.

Table VII-2. Macro-Function Timing Equations for Job 1 and the Dictionary/Directory Processor.

SYNONYM I

STEP	MACRO-FUNCTION	TIMING EQUATION
1	METS	$T_{METS} = P[t_{TRN} + t_{LCM} + t_{EQC}] + (P + 1)[t_{CLR} + t_{XXY}] + t_{SET}$
2	FRTA	$T_{FRTA} = (2)t_{ENR} + t_{FRF} + (\lceil \frac{TA}{31} \rceil)t_{LCA}$
3	CONN	$T_{CONN} = t_{ENR}$
4	MATD	$T_{MATD} = t_{CLR} + (n_2+2)t_{ENR} + (n_2+1)t_{FRF}$
5	SRET	$T_{SRET} = n_2 t_{FRF} + (n_2)(P_1)[t_{LCA} + t_{SCM} + t_{TRN}] + 7n_2[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
6	CONN	$T_{CONN} = t_{ENR}$
7	RELA	$T_{RELA} = (2)t_{ENR}$
8	MFRD	$T_{MFRD} = (n_1+2)t_{ENR} + (n_1+1)t_{FRF} + t_{CLR}$

Table VII-2 (continued)

STEP	MACRO-FUNCTION	TIMING EQUATION
9	MRET	$T_{MRET} = (3n_1)[t_{LCA} + t_{SCM} + t_{TRN}] + (3n_1+1)t_{FRF}$
10	CRET	$T_{CRET} = P[t_{TRN} + t_{SCM}] + t_{CLR}$

SYNONYM II

1	METS	$T_{METS} = P[t_{TRN} + t_{LCM} + t_{EQC}] + (P+1)[t_{CLR} + t_{XXY}] + t_{SET}$
2	FRTA	$T_{FRTA} = (2)t_{ENR} + t_{FRF} + (\lceil \frac{TA}{31} \rceil)t_{LCA}$
3	CONN	$T_{CONN} = t_{ENR}$
4	MATD	$T_{MATD} = t_{CLR} + (n_2+2)t_{ENR} + (n_2+1)t_{FRF}$
5	SRET	$T_{SRET} = n_2 t_{FRF} + (n_2)(P_1)[t_{LCA} + t_{SCM} + t_{TRN}] + 7n_2[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
6	CONN	$T_{CONN} = t_{ENR}$

Table VII-2 (continued)

STEP	MACRO-FUNCTION	TIMING EQUATION
7	SFRD	$T_{SFRD} = (2)t_{ENR} + t_{CLR}$
8	SRET	$T_{SRET} = (3)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
9	CRET	$T_{CRET} = P[t_{TRN} + t_{SCM}] + t_{CLR}$

Once they are found they are retrieved. Then if the data base has Type I synonyms, all (n_1) of the F/R's shortened names and descriptors that are related to the given F/R are required. This is necessary because to obtain the occurrences of some attributes in the given F/R, its related F/Rs may have to be accessed. The process must then multiply retrieve all the data for the related F/Rs and the given F/R name. If the data base has Type II synonyms then only the descriptors of the given F/R need to be found and retrieved.

The first six steps shown in Fig. VII-1 and Table VII-2 for Job 1 are identical for synonyms Type I and II. These are as follows:

Step 1) A multiple (P) word equal to search is performed to find the given F/R name in AM3;

Step 2) Find all the attribute's shortened names in AM2 that are associated with the F/R name in AM3;

Step 3) Find all the attribute names in AM1 of all the attribute shortened names found in AM2;

Step 4) For each attribute (n_2), one at a time find their descriptors in the RAM/AM;

Step 5) Before finding the next attribute's descriptors (Step 4 again), retrieve the attribute's name, shortened name, and its six descriptors;

Step 6) To find the given F/R's shortened name requires the connection of AM3 and AM4.

If the DBMS has Type I synonyms, then the following steps are performed:

Step 7) Find all the F/R names and shortened names in AM3 and AM4 that are related to the F/R name found in AM3 by Step 1;

Step 8) For each (n_1) F/R, find its descriptors;

Step 9) Before finding the next F/R's descriptors (Step 8 again), retrieve the F/R's shortened name and its descriptors;

Step 10) Before ending the job, the given F/R name stored in the comparand register of AM3 must be retrieved.

If the DBMS has Type II synonyms, then the following steps are performed:

Step 7) For the given F/R name, find all its descriptors;

Step 8) For the given F/R name, retrieve all its descriptors and shortened name;

Step 9) Before ending the job, the given F/R name stored in the comparand register of AM3 must be retrieved.

To obtain the equation T_{1I}, which expresses the total time to perform Job 1 for Type I synonyms, the times for each of the macro-functions must be summed. This is accomplished by summing the timing equations shown in Table VII-2 for synonym I.

$$T_{1I} = T_{METS} + T_{FRTA} + T_{CONN} + T_{MATD} + T_{SRET} + T_{CONN} + T_{RELA} + T_{MFRD} + T_{MRET} + T_{CRET}$$

$$T_{1I} = (P)t_{LCM} + (P)t_{EQC} + [P+1]t_{XXY} + t_{SET} + [4n_1 + 9n_2 + 4]t_{FRF} + [n_2(P_1+7) + \left\lceil \frac{TA}{31} \right\rceil + 3n_1]t_{LCA} + [P+4]t_{CLR} + [n_2(P_1+7) + 3n_1 + P]t_{SCM} + [2P + n_2(P_1+7) + 3n_1]t_{TRN} + [n_1 + n_2 + 10]t_{ENR}.$$

To obtain the equation for T_{1II}, the total time to perform Job 1 for Type II synonyms, the times for each of the macro-functions must be summed. This is accomplished by summing the timing equations shown in Table VII-2 for synonym II.

$$T_{1II} = T_{METS} + T_{FRTA} + T_{CONN} + T_{MATD} + T_{SRET} + T_{CONN} + T_{SFRD} + T_{SRET} + T_{CRET}$$

$$T_{1II} = (P)t_{LCM} + (P)t_{EQC} + [P+1]t_{XXY} + t_{SET} + [9n_2 + 5]t_{FRF} + [3 + \left\lceil \frac{TA}{31} \right\rceil + n_2(P_1+7)]t_{LCA} + [P + 4]t_{CLR} + [n_2(P_1+7) + 3 + P]t_{SCM} + [2P + n_2(P_1+7) + 3]t_{TRN} + [n_2 + 8]t_{ENR}.$$

JOB 2

The second job is for the DBMS to provide a subset of the occurrences of an F/R given the F/R name and some operation on

a number (n_5) of its attributes. The development of the timing equation for this job is similar to the development for the first job. Consider Fig. VII-2 and Table VII-3. The first operation is to perform n_5 ETS's for each of the attribute names given to the DBMS. This is followed by a determination of each of their shortened names and the F/R names (n_4) with which all n_5 attributes are associated. The next operation is to determine the descriptors of the n_5 attributes and to retrieve the attribute names, shortened names, and their descriptors. The DDP then retrieves all (n_4) the F/R names that contain the n_5 attributes. If one of these names is the F/R name given in the query to the DBMS, then the process continues by performing an ETS on the F/R name. The rest of Job 2 proceeds identically to Job 1 (i.e. from Job 1, step 6 onward).

The first eight steps shown in Fig. VII-2 and Table VII-3 for Job 2 are identical for synonyms Type I and II. These are as follows:

Step 1) (n_5) multiple (P_1) equal to searches are performed to find the given attribute names in AM1;

Step 2) Find all the given (n_5) attribute names' shortened names by interacting AM1 to AM2;

Step 3) Find the F/R(s) (n_4) in which all the given n_5 attributes are associated by interacting AM2 to AM3 using ARRAY III;

Step 4) For each attribute (n_5), one at a time, find their six descriptors in the RAM/AM;

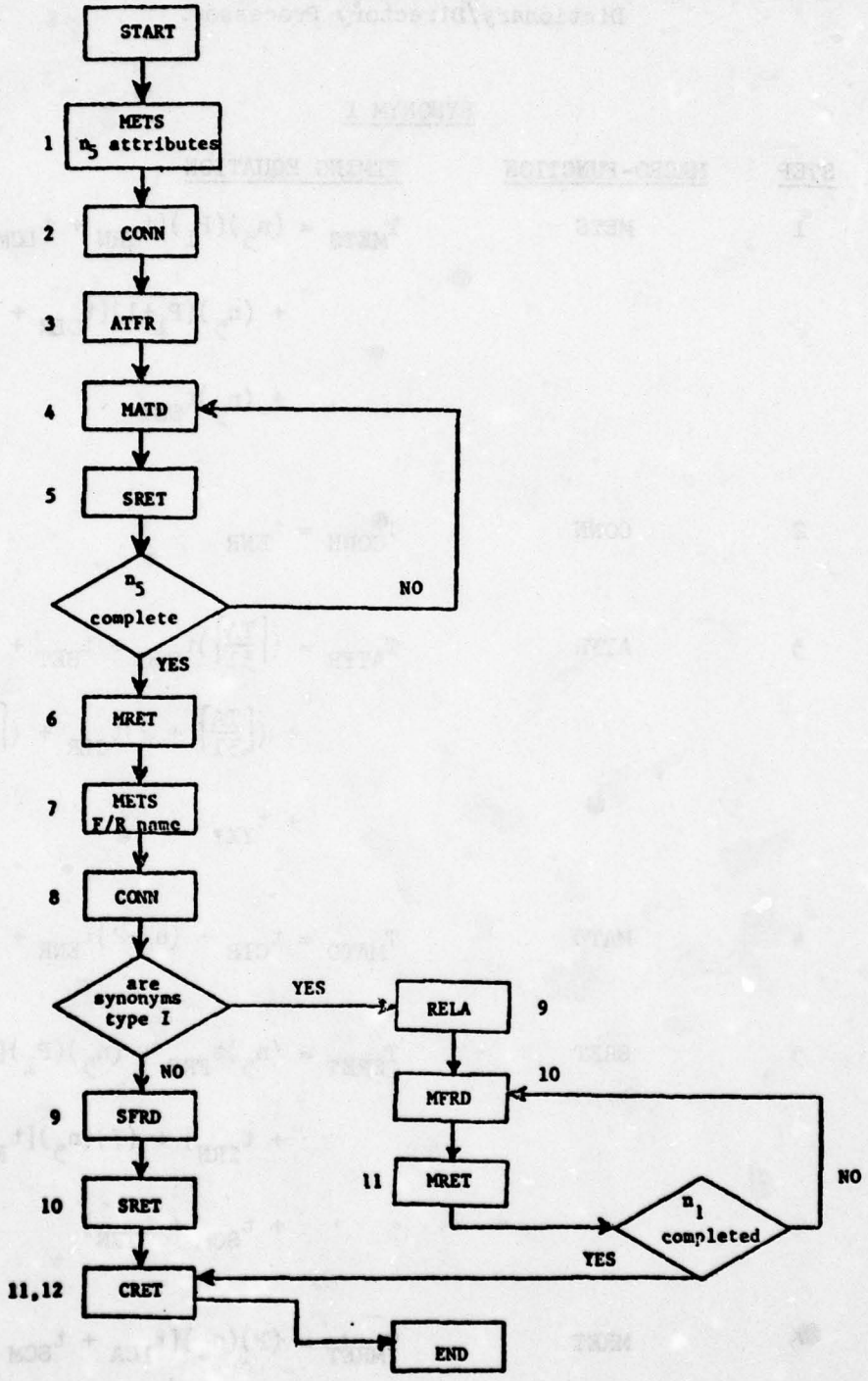


Figure VII-2. Job 2 Flow Diagram for the Dictionary/Directory Processor.

Table VII-3. Macro-Function Timing Equations for Job 2 and the Dictionary/Directory Processor.

SYNONYM I

STEP	MACRO-FUNCTION	TIMING EQUATION
1	METS	$T_{METS} = (n_5)(P_1)[t_{TRN} + t_{LCM} + t_{EQC}] + (n_5)(P_1+1)[t_{CLR} + t_{XXY}] + (n_5)t_{SET}$
2	CONN	$T_{CONN} = t_{ENR}$
3	ATFR	$T_{ATFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 2)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}$
4	MATD	$T_{MATD} = t_{CLR} + (n_5+2)t_{ENR} + (n_5+1)t_{FRF}$
5	SRET	$T_{SRET} = (n_5)t_{FRF} + (n_5)(P_1)[t_{LCA} + t_{SCM} + t_{TRN}] + (7)(n_5)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
6	MRET	$T_{MRET} = (P)(n_4)[t_{LCA} + t_{SCM} + t_{TRN}] + (n_4+1)t_{FRF}$

Table VII-3 (continued)

STEP	MACRO-FUNCTION	TIMING EQUATION
7	METS	$T_{METS} = (P)[t_{TRN} + t_{LCM} + t_{EQC}] + (P+1)[t_{CLR} + t_{XXY}] + t_{SET}$
8	CONN	$T_{CONN} = t_{ENR}$
9	RELA	$T_{RELA} = (2)t_{ENR}$
10	MFRD	$T_{MFRD} = (n_1+2)t_{ENR} + (n_1+1)t_{FRF} + t_{CLR}$
11	MRET	$T_{MRET} = (3)(n_1)[t_{LCA} + t_{SCM} + t_{TRN}] + (3n_1 + 1)t_{FRF}$
12	CRET	$T_{CRET} = P[t_{TRN} + t_{SCM}] + t_{CLR}$
		SYNONYM II
1	METS	$T_{METS} = (n_5)(P_1)[t_{TRN} + t_{LCM} + t_{EQC}] + (n_5)(P_1+1)[t_{CLR} + t_{XXY}] + (n_5)t_{SET}$
2	CONN	$T_{CONN} = t_{ENR}$

Table VII-3 (continued)

STEP	MACRO-FUNCTION	TIMING EQUATION
3	ATFR	$T_{ATFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 2)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}$
4	MATD	$T_{MATD} = t_{CLR} + (n_5+2)t_{ENR} + (n_5+1)t_{FRF}$
5	SRET	$T_{SRET} = (n_5)t_{FRF} + (n_5)(P_1)[t_{LCA} + t_{SCM} + t_{TRN}] + (7)(n_5)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
6	MRET	$T_{MRET} = (P)(n_4)[t_{LCA} + t_{SCM} + t_{TRN}] + (n_4 + 1)t_{FRF}$
7	METS	$T_{METS} = (P)[t_{TRN} + t_{LCM} + t_{EQC}] + (P+1)[t_{CLR} + t_{XXY}] + t_{SET}$
8	CONN	$T_{CONN} = t_{ENR}$
9	SFRD	$T_{SFRD} = (2)t_{ENR} + t_{CLR}$

Table VII-3 (continued)

STEP	MACRO-FUNCTION	TIMING EQUATION
10	SRET	$T_{SRET} = (3)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
11	CRET	$T_{CRET} = P[t_{TRN} + t_{SCM}] + t_{CLR}$

Step 5) Before finding the next attribute's descriptors (Step 4 again), retrieve the attribute's name, shortened name and its six descriptors;

Step 6) Before proceeding--the names of the (n_4) F/Rs need to be retrieved from AM3. If one of them is equivalent to the given F/R name, then the job will continue;

Step 7) A multiple (P) word equal to search is performed to find the given F/R name in AM3;

Step 8) To find the given F/R's shortened name requires the interaction of AM3 to AM4.

If the DBMS has Type I synonyms, then the following are performed:

Step 9) Find all the F/R names and shortened names in AM3 and AM4 that are related to the F/R name found in AM3 by step 7;

Step 10) For each (n_1) F/R, find its descriptors;

Step 11) Before finding the next F/R's descriptors (Step 10 again), retrieve the F/R's shortened name and its descriptors;

Step 12) Before ending the job, the given F/R name stored in the comparand register of AM3 must be retrieved.

If the DBMS has Type II synonyms, then the following steps are performed:

Step 9) For the given F/R name, find all its descriptors;

Step 10) For the given F/R name, retrieve all its descriptors and shortened name;

Step 11) Before ending the job, the given F/R name stored in the comparand register of AM3 must be retrieved.

To obtain the equation T_{2I} which expresses the total time to perform Job 2 for Type I synonyms, the times for each of the macro-functions must be summed. This is accomplished by summing the timing equations shown in Table VII-3 for synonym I.

$$T_{2I} = T_{METS} + T_{CONN} + T_{ATFR} + T_{MATD} + T_{SRET} + T_{MRET}$$
$$+ T_{METS} + T_{CONN} + T_{RELA} + T_{MFRD} + T_{MRET} + T_{CRET}$$

$$T_{2I} = [n_5P_1+P]t_{LCM} + [\lceil \frac{TA}{31} \rceil + n_5P_1+P]t_{EQC}$$
$$+ [n_5(P_1+1) + \lceil \frac{TA}{31} \rceil + P + 1]t_{XXY} + [n_5 + 2]t_{SET}$$
$$+ [9n_5 + n_4 + 4n_1 + 4]t_{FRF} + [n_5(P_1+7) + P\,n_4 + 3n_1]t_{LCA}$$
$$+ [n_5(P_1+1) + \lceil \frac{TA}{31} \rceil + P + 6]t_{CLR}$$
$$+ [n_5(P_1+7) + Pn_4 + 3n_1 + P]t_{SCM}$$
$$+ [n_5(2P_1+7) + Pn_4 + 3n_1 + 2P]t_{TRN} + t_{YXY}$$
$$+ [n_5 + n_1 + 12]t_{ENR}.$$

To obtain the equation T_{2II}, the total time to perform Job 2 for Type II synonyms, the times for each of the macro-functions

must be summed. This is accomplished by summing the timing equations shown in Table VII-3 for synonym II.

$$T_{2II} = T_{METS} + T_{CONN} + T_{ATFR} + T_{MATD} + T_{SRET} + T_{MRET}$$
$$+ T_{METS} + T_{CONN} + T_{SFRD} + T_{SRET} + T_{CRET}$$

$$T_{2II} = [n_5P_1+P]t_{LCM} + [\lceil \frac{TA}{31} \rceil + n_5P_1 + P]t_{EQC}$$
$$+ [n_5(P_1+1) + \lceil \frac{TA}{31} \rceil + P + 1]t_{XXY} + [n_5 + 2]t_{SET}$$
$$+ [9n_5 + n_4 + 5]t_{FRF} + [n_5(P_1+7) + Pn_4 + 3]t_{LCA}$$
$$+ [n_5(P_1+1) + P + 6 + \lceil \frac{TA}{31} \rceil]t_{CLR}$$
$$+ [n_5(P_1+7) + P(n_4+1) + 3]t_{SCM}$$
$$+ [n_5(2P_1+7) + P(n_4+2) + 3]t_{TRN} + t_{YXY}$$
$$+ [n_5 + 10]t_{ENR}.$$

JOB 3

The third generic job for the DBMS is to modify all the occurrences of an attribute in a data base with Type II synonyms. This requires the DBMS to modify all the F/Rs which have an association with either the attribute in question or any of its synonyms. The process is shown in Fig. VII-3 and Table VII-4. The process assumes the attribute shortened name has been previously determined, therefore the first step is to perform an ETS on the attribute's shortened name. The next step is to

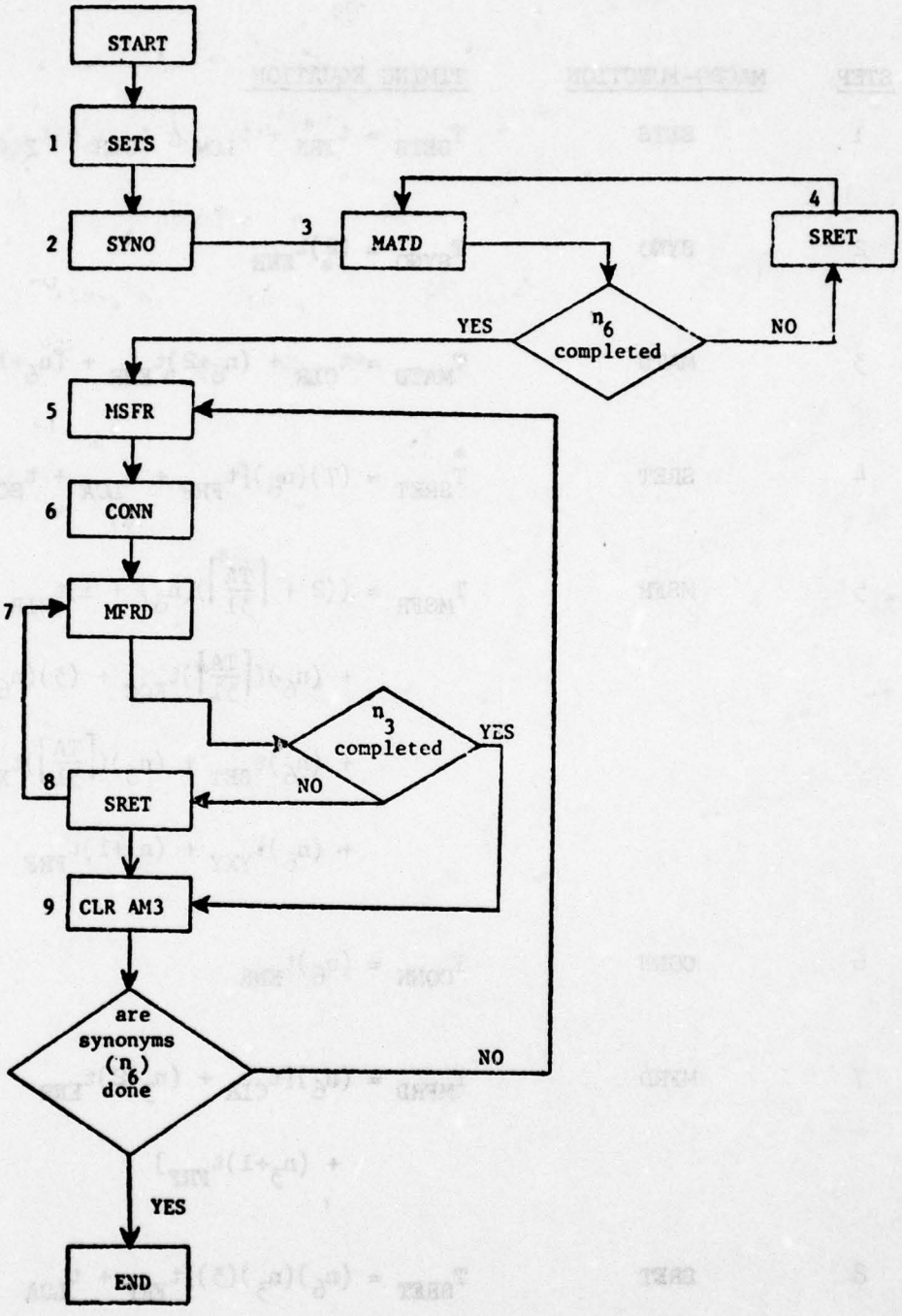


Figure VII-3. Job 3 Flow Diagram for the Dictionary/ Directory Processor.

Table VII-4. Macro-Function Timing Equations for Job 3 and the Dictionary/Directory Processor.

STEP	MACRO-FUNCTION	TIMING EQUATION
1	SETS	$T_{SETS} = t_{TRN} + t_{LCM} + t_{CLR} + t_{EQC}$
2	SYNO	$T_{SYNO} = (2)t_{ENR}$
3	MATD	$T_{MATD} = t_{CLR} + (n_6+2)t_{ENR} + (n_6+1)t_{FRF}$
4	SRET	$T_{SRET} = (7)(n_6)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
5	MSFR	$T_{MSFR} = ((2 + \lceil \frac{TA}{31} \rceil)(n_6) + 1)t_{CLR} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{EQC} + (3)(n_6 + \frac{2}{3})t_{ENR} + (n_6)t_{SET} + (n_6)(\lceil \frac{TA}{31} \rceil)t_{XXY} + (n_6)t_{YXY} + (n_6+1)t_{FRF}$
6	CONN	$T_{CONN} = (n_6)t_{ENR}$
7	MFRD	$T_{MFRD} = (n_6)[t_{CLR} + (n_3+2)t_{ENR} + (n_3+1)t_{FRF}]$
8	SRET	$T_{SRET} = (n_6)(n_3)(3)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRM}]$
9	CLR	$T_{CLR} = (n_6)t_{CLR}$

determine all (n_6) of its synonyms and retrieve all of their shortened names and descriptors. Then each attribute shortened name is used to find its associated (n_3) F/Rs. Then the F/R shortened names are retrieved with their descriptors. This is followed by retrieving all n_3 F/R shortened names, their descriptors, and the clearing of AM3 so that the next synonym attribute can be processed.

A detailed description of this job can be obtained from the following steps describing Fig. VII-3 and Table VII-4:

Step 1) Perform an equal to search on the given attribute's shortened name using AM2;

Step 2) Determine the given attribute's synonyms by utilizing ARRAY I and AM1. This will provide all the attribute's names and shortened names of its synonyms;

Step 3) For each attribute (n_6), one at a time, find their six descriptors in the RAM/AM;

Step 4) Before finding the next attribute's descriptors (Step 3 again), retrieve the attribute's shortened name and its six descriptors;

Step 5) For each synonym attribute name and shortened name (n_6), one at a time, find its associated F/R names (n_3) by using ARRAY III to associate AM1 and AM2 with AM3;

Step 6) Before performing Step 5 again, the resultant F/R names from Step 5 are to be interacted to

their shortened names in AM4 and then Step 7 would be performed;

Step 7) For each F/R (n_3), one at a time, find their two descriptors in the RAM/AM;

Step 8) Before finding the next F/R's descriptors (Step 7 again), retrieve the F/R's shortened name and its two descriptors. If all n_3 F/R's are complete go to Step 9;

Step 9) Clear the response register of AM3 and proceed to Step 5. If all n_6 attributes have been processed, the job is ended.

To obtain the equation T_{3II}, the total time to perform Job 3 for Type II synonyms, the time for each of the macro-functions, must be summed. This is accomplished by summing the timing equations shown in Table VII-4.

$$T_{3II} = T_{SETS} + T_{SYNO} + T_{MATD} + T_{SRET} + T_{MSFR} + T_{CONN} + T_{MFRD} + T_{SRET} + (n_6)\, T_{CLR}.$$

$$T_{3II} = t_{LCM} + [n_6(\lceil \tfrac{TA}{31} \rceil) + 1]t_{EQC} + [n_6(\lceil \tfrac{TA}{31} \rceil)]t_{XXY} + [n_6]t_{SET} + [n_6(10+4n_3) + 2]t_{FRF} + [n_6(7+n_3)]t_{LCA} + [n_6(4 + \lceil \tfrac{TA}{31} \rceil) + 3]t_{CLR} + [n_6(7+3n_3)]t_{SCM} + [n_6(7+3n_3) + 1]t_{TRN} + [n_6]t_{YXY} + [n_6(n_3+7) + 6]t_{ENR}.$$

JOB 4

The fourth, and last, generic job is to modify all the occurrences of an attribute in a data base with Type I synonyms. This requires the DBMS to modify all the F/Rs which have an association with the attribute in question's stored synonym. The process is shown in Fig. VII-4 and Table VII-5. As discussed in Job 3 above, the attribute's shortened name is used in an ETS. This is followed by locating the stored synonym's descriptors. Then the first attribute shortened name (i.e. the stored synonym's shortened name) is used to determine all associated (n_3) F/R names. Then the process finds and retrieves all (n_3) of the F/R's shortened names and their descriptors. The last step retrieves the stored attribute's shortened name, its descriptors, and clears AM1's response register.

A detailed description of this job can be obtained from the following steps describing Fig. VII-4 and Table VII-5:

Step 1) Perform an equal to search on the given attribute's shortened name using AM2;

Step 2) Determine the given attribute's stored synonym and its descriptors;

Step 3) From Step 2 only the stored synonym is respondent in AM2, therefore find all (n_3) the F/R's in which it is associated by using ARRAY III to interact with AM3;

Step 4) Find all (n_3) of the shortened names of the F/Rs found in Step 3 by interacting AM3 to AM4;

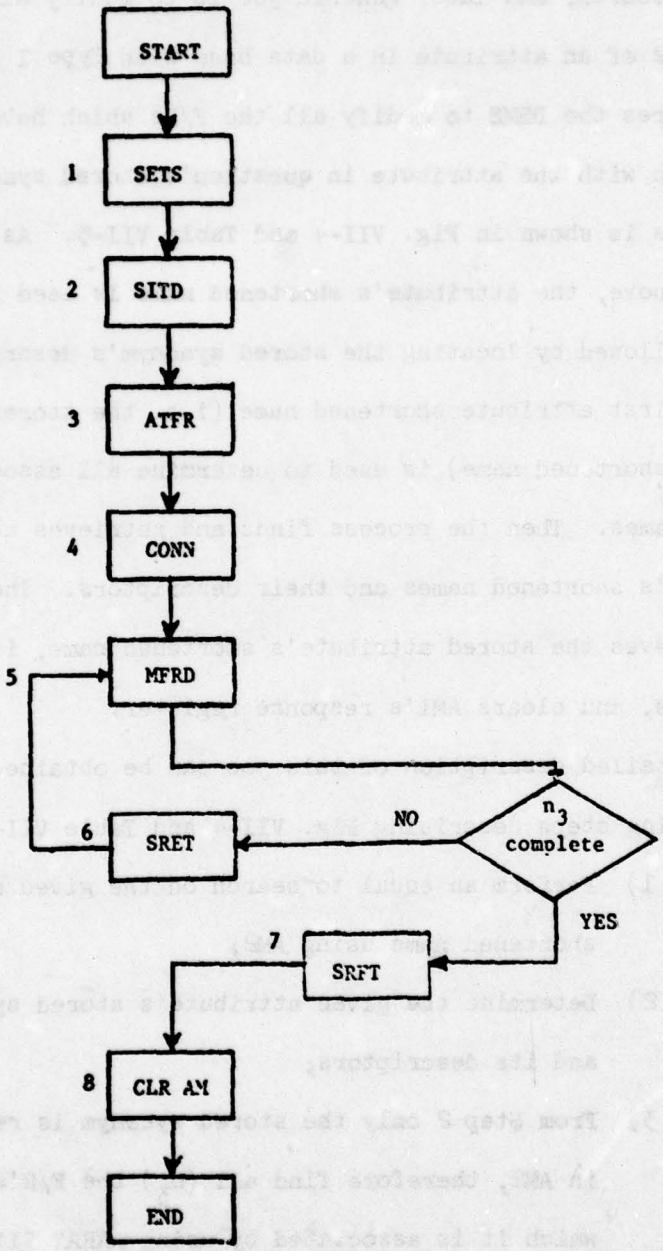


Figure VII-4. Job 4 Flow Diagram for the Dictionary/ Directory Processor.

Table VII-5. Macro-Function Timing Equations for Job 4 and the Dictionary/Directory Processor.

STEP	MACRO-FUNCTION	TIMING EQUATION
1	SETS	$T_{SETS} = t_{TRN} + t_{LCM} + t_{CLR} + t_{EQC}$
2	SITD	$T_{SITD} = (3)t_{ENR} + (2)t_{CLR} + t_{FRA}$
3	ATFR	$T_{ATFR} = (\lceil \frac{TA}{31} \rceil)t_{EQC} + t_{SET} + (4)t_{ENR} + (\lceil \frac{TA}{31} \rceil + 2)t_{CLR} + (\lceil \frac{TA}{31} \rceil)t_{XXY} + t_{YXY}$
4	CONN	$T_{CONN} = t_{ENR}$
5	MFRD	$T_{MFRD} = (n_3+2)t_{ENR} + (n_3+1)t_{FRF} + t_{CLR}$
6	SRET	$T_{SRET} = (7)[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
7	SRET	$T_{SRET} = 3n_3[t_{FRF} + t_{LCA} + t_{SCM} + t_{TRN}]$
8	CLR	$T_{CLR} = t_{CLR}$

Step 5) For each F/R name and shortened name, one at a time, find their two descriptors by interacting ARRAY IV to the RAM/AM;

Step 6) Before finding the next F/R's descriptors (Step 5 again), retrieve the F/R's shortened name and its two descriptors;

Step 7) Retrieve the stored synonym's attribute shortened name and its six descriptors from AM2 and the RAM/AM, respectively;

Step 8) Clear the response register of AM1.

To obtain the equation T_{4I}, the total time to perform Job 4 for Type I synonyms, the time for each of the macro-functions must be summed. This is accomplished by summing the timing equations shown in Table VII-5.

$$T_{4I} = T_{SETS} + T_{SITD} + T_{ATFR} + T_{CONN} + T_{MFRD} + T_{SRET} + T_{SRET} + T_{CLR}$$

$$T_{4I} = t_{LCM} + [\lceil \frac{TA}{31} \rceil + 1]t_{EQC} + [\lceil \frac{TA}{31} \rceil]t_{XXY} + t_{SET} + [4n_3 + 8]t_{FRF} + [3n_3 + 7]t_{LCA} + [\lceil \frac{TA}{31} \rceil + 7]t_{CLR} + [3n_3 + 7]t_{SCM} + [3n_3 + 8]t_{TRN} + t_{YXY} + [n_3 + 10]t_{ENR} + t_{FRA}.$$

This concludes the development of the timing equations for the DDP. These equations are general and can be evaluated for different hardware capabilities, i.e. transfer rate, degree of parallelism, etc. The actual timings appearing at the end of

this chapter are for those timings of the micro-functions that were obtained from the STARAN. If different timings are available then they could be used in the above derived equations.

SEQUENTIAL COMPUTER IMPLEMENTATION

The next portion of this chapter describes the development of the mathematical equations that can be used to determine the time to perform the same generic jobs on a sequential machine. Like the DDP discussed above, the MIX computer's software directs two basic tasks in the performance of the generic jobs. The first task is to perform an equal to search (ETS) and the second task is to transfer the data found with the "successful" ETS to the user's area in the main memory of the MIX computer.

To perform the first task, it was necessary to develop a fast, efficient, and realistic ETS technique. Two techniques were studied. These were hashing and explicit binary tree searching. Appendix B provides a discussion of both techniques and presents the data showing how much faster a hashing technique is than an explicit binary search technique. A hashing function can be divided into two parts. The first part is its hashing part and its second part is its collision resolution part. The first part functions on the value of the key and based on its current value the hashing technique generates a memory (or peripheral) location. If the contents of this location are not equivalent to the interested key value; then the collision resolution part of the hashing technique must resolve the conflict.

Since this problem deals with a data dictionary and a partial data directory, the keys to be hashed are F/R names and attribute names. These keys are unique, finite, and to a certain extent, do not change their names very often. Therefore, it is assumed that a hashing technique could be obtained that generated no collisions.

This assumption allows the timing results for the sequential computer to be extremely fast. It also implies that the algorithm must be capable of accommodating some minor changes due to additions, deletions, and changes to F/R and attribute names. However, if a data base has many changes in names, number of F/Rs and attributes, then this assumption should be discarded and a model should be developed to represent the additional time required to handle collisions.

The timing equations for the hashing and the transfer of data are developed by writing the software code in MIXAL, MIX's assembly language, and then summing the number of times each statement or operation is executed multiplied by the time required to perform its operation divided by U. This provides a normalized time per MIXAL statement, i.e.

NTS = (no. of times an operation is executed)(time units/ operation) ÷ U.

The NTS for each statement is summed and then multiplied by U which is a unit of time or relative measure (e.g. μseconds) described by Knuth [24], such that

> . . .ADD, SUB, all LOAD operations, all STORE operations (including STZ), all shift commands and all comparison operations take two units of time. MOVE requires one unit plus two for each word moved. MUL requires 10 and

DIV requires 12 units. Execution time for floating-point operations is unspecified. All remaining operations take one unit of time, plus the time the computer may idle on the IN, OUT, IOC, or HLT instructions.

Therefore the total time for a MIXAL code is

$$\left(\sum_{\forall \text{ statements}} \text{NTS} \right) \text{U}.$$

HASHING TECHNIQUE

The hashing technique chosen for the MIX computer is a simple division method where the key value is divided by a value M and the remainder is used as the location (or address) of where the key is stored. The derivation of the total time for keys of one computer word long will be developed first. This will be followed by similar developments for keys whose sizes are two computer words and three computer words in length. From these times a general equation for key sizes ≥ 2 computer words will be developed.

For the development of the hashing timing equations, let M be the key's dividing value, let L be a dummy location in the memory, and let S be the smallest possible hashed key value so that when subtracted from the remainder it normalizes the locations to the zero location. The S variable emulates some control over where the key values are stored. The other important point in understanding the following codes is that stored in the hashed location is the address of where the key value is stored. This allows the data affiliated with each key value to

be stored with the key value and therefore not interfering with the hashing locations of other key values. Shown in Fig. VII-5 is an example of the memory layout being discussed. The key values to be hashed on are stored in locations k_1, k_2, and k_3. These values, after being hashed, provide a location value between location 0 and RST-1. The contents of these locations contain the location of the key value followed by its affiliated data. Examples of these locations are RST and UXZ.

Keys of One Computer Word Long

Consider the following MIXAL code for hashing a key that is one computer word long.

Line No.	Location	Operation	Address	NTS	Explanation
01		LDX	K	2	Load the searching key value into register X, rX
02		ENTA	0	1	Sets rA = 0
03		DIV	=M=	12	Divides rA and rX by M
04		SLAX5		2	Shifts the remainder in rX into rA
05		SUB	=S=	2	Subtract from the remainder the lowest possible address
06		STA	L	2	Stores the result of line 5 into location L
07		LD1	L	2	Load the contents of location L into r1
08		LDA	0,1	2	Loads the address of the hashed key value in rA

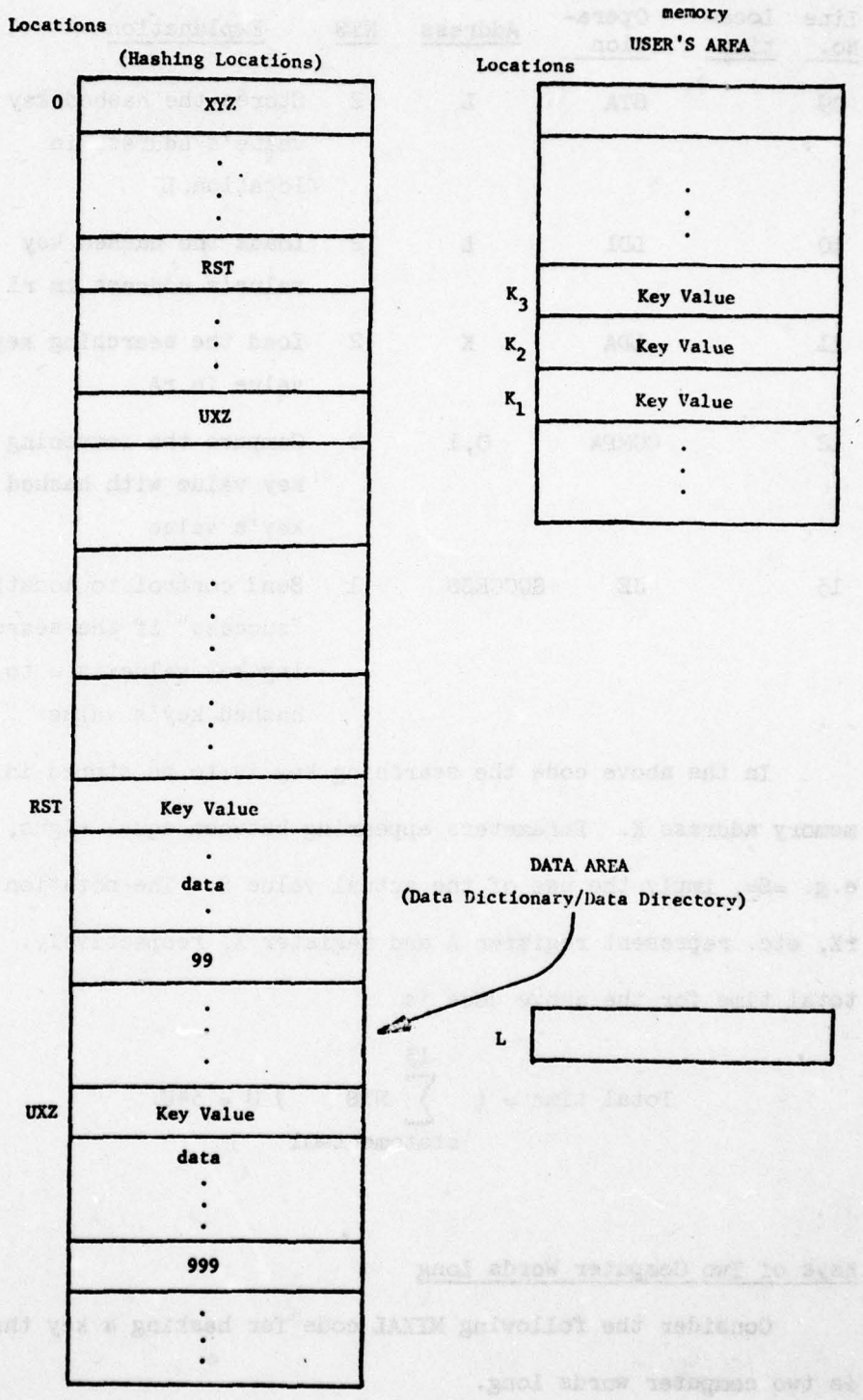


Figure VII-5. Sample Sequential Memory Data Arrangement.

Line No.	Location	Operation	Address	NTS	Explanation
09		STA	L	2	Stores the hashed key value's address in location L
10		LD1	L	2	Loads the hashed key value's address in r1
11		LDA	K	2	Load the searching key value in rA
12		COMPA	0,1	2	Compare the searching key value with hashed key's value
13		JE	SUCCESS	1	Send control to location "success" if the searching key value is = to the hashed key's value

In the above code the searching key is to be stored in memory address K. Parameters appearing between equal signs, e.g. =S=, imply the use of the actual value S. The notation rA, rX, etc. represent register A and register X, respectively. The total time for the above code is

$$\text{Total time} = \left(\sum_{\text{statement}=01}^{13} \text{NTS} \right) U = 34U.$$

Keys of Two Computer Words Long

Consider the following MIXAL code for hashing a key that is two computer words long.

Line No.	Location	Operation	Address	NTS	Explanation
01		LDX	K1	2	Load 2nd half of searching key into rX
02		LDA	K2	2	Load 1st half of searching key into rA
03		DIV	=M=	12	Divide rA and rX by M
04		SLAX5		2	Shifts the remainder in rX into rA
05		SUB	=S=	2	Subtract from the remainder the lowest possible address
06		STA	L	2	Stores the result of line 5 into location L
07		LD1	L	2	Load the contents of location L into r1
08		LDA	0,1	2	Loads the address of the hashed key value into rA
09		STA	L	2	Stores the hashed key value's address in location L
10		LD1	L	2	Loads the hashed key value's address in r1
11		LDA	K2	2	Load 1st half of the searching key value in rA
12		COMPA	0,1	2	Compare 1st half of searching key value with 1st half of hashed key's value
13		JNE	BEND	1	If they are not equal, jump to bad end (BEND)
14		LDA	K1	2	Load 2nd half of searching key value in rA
15		COMPA	1,1	2	Compare 2nd half of searching key value with 2nd half of hashed key's value
16		JE	SUCCESS	1	Jump, if they are equal, to SUCCESS
17	BEND				

In the above code, the "higher" portion of the searching key is assumed to be stored in memory location K2 and the "lower" portion is assumed to be stored in memory location K1. The concept of "higher" and "lower" portions represent the highest significant bits and lowest significant bits if the key was represented as a binary integer. For example, if the word CAT were converted to a binary integer, the C would have the highest significant bits and T would have the lowest significant bits.

The total time for the above code is

$$\text{Total time } \left(\sum_{\text{Statement 01}}^{17} \text{NTIS} \right) U = 40U.$$

Keys of Three Computer Words Long

Consider the following MIXAL code for hashing a key that is three computer words long.

Line No.	Location	Operation	Address	NTS	Explanation
01		LDX	K2	2	Load middle 3rd of searching key into rX
02		LDA	K3	2	Load highest 3rd of searching key into rA
03		DIV	=M=	12	Divide rA and rX by M
04		SLAX5		2	Shifts the remainder in rX into rA
05		LDX	K1	2	Load lowest 3rd of searching key into rX
06		DIV	=M=	12	Divide rA and rX by M
07		SLAX5		2	Shifts the remainder in rX into rA

Line No.	Location	Operation	Address	NTS	Explanation
08		SUB	=S=	2	Subtract from the remainder the lowest possible address
09		STA	L	2	Stores the result of line 8 into location L
10		LD1	L	2	Load the contents of location L in r1
11		LDA	0,1	2	Loads the address of the hashed key value into rA
12		STA	L	2	Stores the hashed key value's address in location L
13		LD1	L	2	Loads the hashed key values address in r1
14		LDA	K3	2	Load the highest 3rd of the searching key value in rA
15		COMPA	0,1	2	Compare the highest 3rd of the searching key value with the highest 3rd of the hashed key's value
16		JNE	BEND	1	If they are not equal, jump to bad end (BEND)
17		LDA	K2	2	Load middle 3rd of the searching key value in rA
18		COMPA	1,1	2	Compare the middle 3rd of the searching key value with the middle 3rd of the hashed key's value
19		JNE	BEND	1	If they are not equal, jump to Bad End (BEND)
20		LDA	K1	2	Load lowest 3rd of the searching key value in rA
21		COMPA	2,1	2	Compare lowest 3rd of the searching key value with lowest 3rd of the hashed key's value
		JE	SUCCESS		Jump, if they are equal, to SUCCESS
22	BEND				

In the above code, the higher portion of the searching key is assumed to be stored in memory location K3; the middle 3rd of the searching key is assumed to be stored in memory location K2; and the lowest 3rd of the searching key is assumed to be stored in the memory location K1. The total time for the above code is

$$\text{Total Time } \Big(\sum_{\text{Statement 01}}^{22} \text{NTS} \Big) U = 61U$$

Key Sizes Equal To or Greater Than Two

To develop a general timing equation for keys ≥ 2 computer words long requires a study of the two MIXAL codes for key lengths of two computer words and three computer words. The difference between the two codes is six statements. To convert the code for two computer words to three computer words requires the addition of lines 05, 06, 07, 19, 20, and 21. Lines 05-07 loads the 3rd portion of the key word, divides by M, and shifts the remainder into rA. Lines 19-21 provide a jump to END if the previous portion of the key was not an exact compare, loads rA with the 3rd portion of the key word, and compares this portion with the stored key's value. The total time for these six lines is 21U. For any additional key lengths, say four computer words long, six more lines with the same operations must be added to the MIXAL code. Intuitively, the increase in time for a key of w computer words long, where $w \geq 2$, is $(w - 2)21U$ beyond what it takes in time for a key of two computer words long (40U).

In this section it has been shown that the time to hash on a key one computer word long is 34U. For keys that are two computer words long or greater ($w \geq 2$) the total time is

$$[40 + (w - 2)21]U.$$

This completes the presentation of an implementation and its timing equations for software to find data utilizing a hashing function. The next portion of this chapter deals with the modeling of the data following the stored key values. These are the data that must be transferred (i.e. the second task) from the data dictionary and partial data directory area of the memory to the user's area.

DATA STRUCTURES

This portion of the chapter deals with the structure of the data located in memory affiliated with the stored key values (see Fig. VII-5). The structures are divided into two groups. Group 1 is for type I synonyms and Group 2 is for type II synonyms. Within each group there are two separate structures, one for an F/R name and one for an attribute name. These structures will be described using DP sets.

The structure of the affiliated data with the key values is very important. It dictates for instance the amount of data that must be moved to the user's memory area. Most importantly, the structure directs the DBMS software in finding needed data such as pointers to other data, the order of occurrences of data, and the amount of data that is affiliated with a particular key

Group 1

The structures in Group 1 for an F/R name and an attribute name can be described as two DP sets named G1FR and G1A, respectively. Let

$$\begin{aligned} G1FR = \{&\text{F/R name, F/R shortened name, F/R primary key,} \\ &\text{F/R pointer, } n_2, \ n_1\} \cup P_s^{P+5}(F/RCM) \cup P_s^{P+5+3}(F/RCM) \\ &\cup \ldots \cup P_s^{P+5+(n_1-1)3}(F/RMC) \cup P_s^{P+5+n_1(3)}(AM) \\ &\cup P_s^{P+5+n_1(3)+(P_1+7)}(AM) \cup \ldots \cup P_s^{P+5+n_1(3)+(n_2-1)(P_1+7)}(AM) \\ &\cup P_s^{P+5+n_1(3)+n_2(P_1+7)} \{\text{End of Record}\}, \end{aligned}$$

where F/RCM = {F/R shortened name, F/R pointer, F/R primary key}

and AM = {Attribute name, attribute shortened name, size descriptor, representation technique, synonym descriptor, uniqueness descriptor, password name, privileges name}.

All the variables in the above DP sets are assumed to be one computer word long except the F/R names and attribute names which are P and P_1 words long, respectively. The synonym and uniqueness descriptors of an attribute only specify whether the respective attribute has a synonym and whether the attribute's values are unique. There are $(n_1 + 1)$ F/R pointers. Each pointer signifies the location of the G1FR of a related F/R. These pointers provide a circular linked list of the data affiliated with related F/Rs. The F/RCM DP sets provide the needed data for accessing any of an F/R's related F/Rs without "following"

the linked list. The DBMS software can find the desired F/R data (G1FR) either by using its F/R pointer or its primary key available in F/RCM. The AM DP set is self-explanatory. For an attribute name let

G1A = {Attribute name, attribute shortened name, size descriptor, representation technique, coding function to the only stored synonym, n_6, pointer 1 to stored synonym, pointer 2, pointer 3, ..., pointer n_6, uniqueness descriptor, password name, privileges name, n_3}

$$\cup\ P_s^{9+n_6+P_1}(F/RM) \cup P_s^{9+n_6+P_1+(P+3)}(F/RM) \cup \ldots$$

$$\cup\ P_s^{9+n_6+P_1+(n_3-1)(P+3)}(F/RM) \cup P_s^{9+n_6+P_1+n_3(P+3)}$$

{End of Record},

where the DP set, F/RM = {F/R name, F/R shortened name, F/R primary key, F/R pointer}.

The pointers one through n_6 are the locations or pointers to the G1A structures of the synonyms of the attribute in question. The first pointer in each G1A, as shown in Fig. VII-6, point to the G1A structure of the attribute whose occurrences have the stored values. The F/RM DP set is self explanatory.

Group 2

The structures in Group 2 for an F/R name and an attribute name can be described as DP sets named G2FR and G2A, respectively.

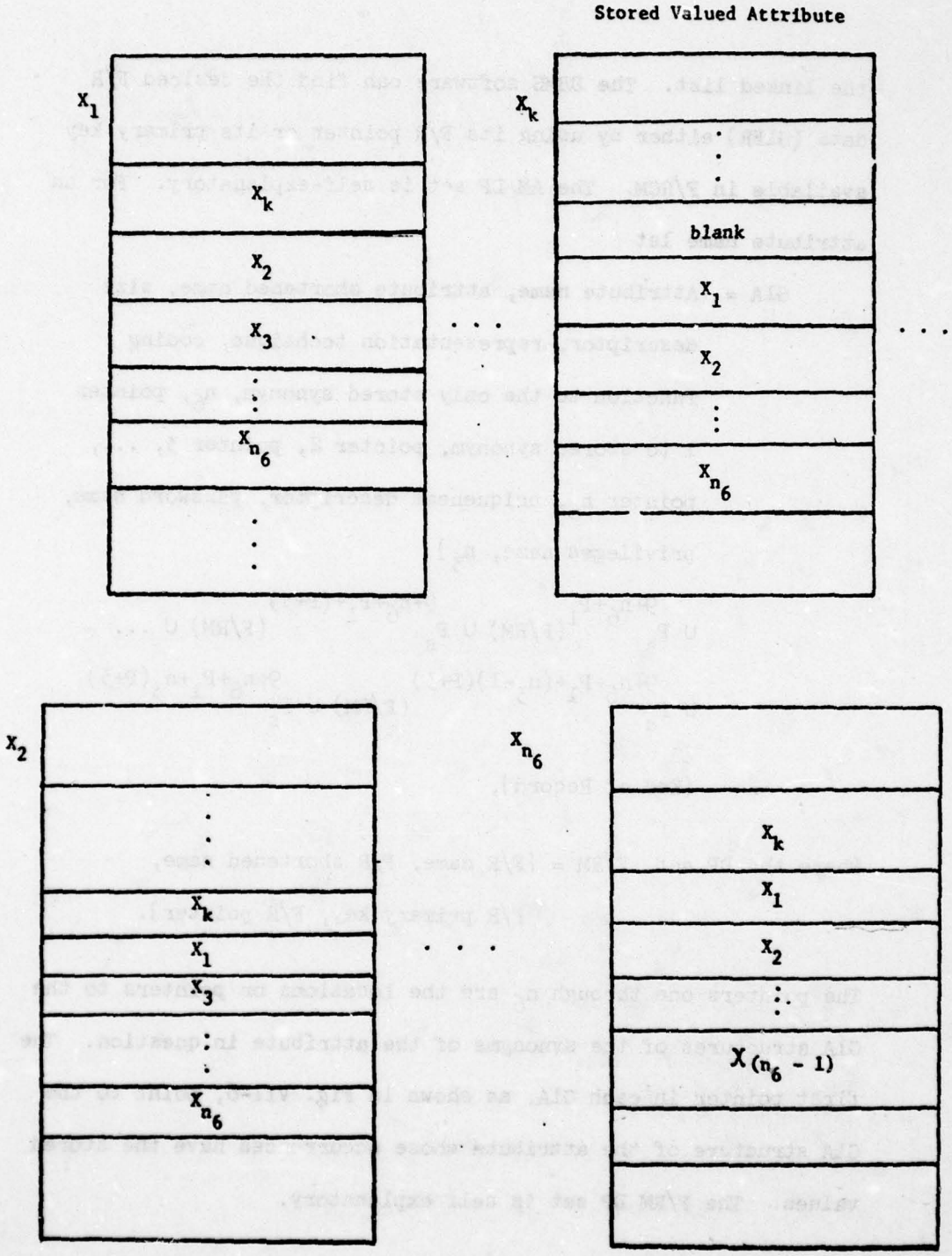
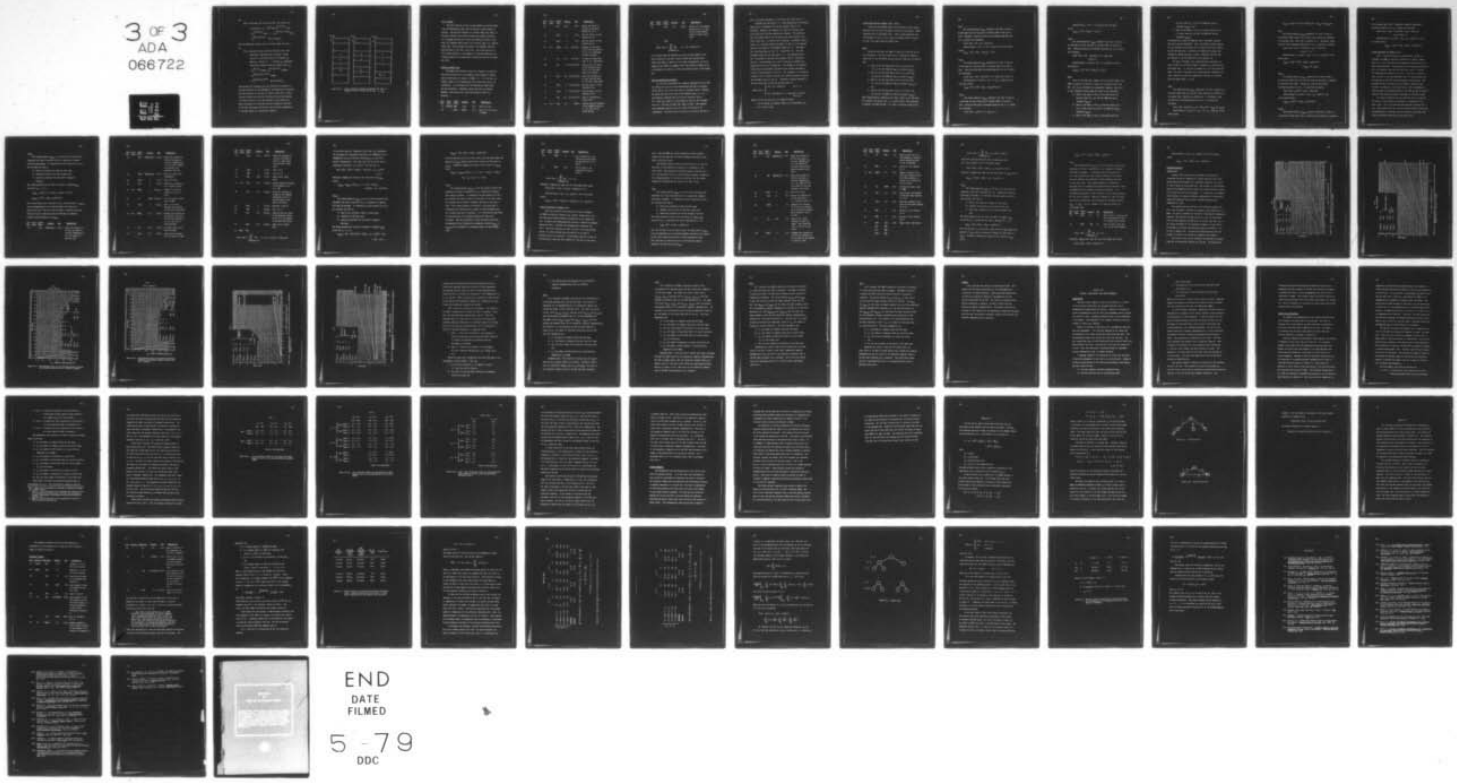


Figure VII-6. Group 1 Attribute Memory Arrangement for a Type I Synonym Stored in Location X_k.

AD-A066 722 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB N Y F/G 9/2
A DATA BASE MANAGEMENT MODELING TECHNIQUE AND SPECIAL FUNCTION --ETC(U)
JAN 79 G T CAPRARO, P B BERRA

UNCLASSIFIED RADC-TR-79-14 NL

3 OF 3
ADA
066722



END
DATE
FILMED
5-79
DDC

Let

$$G2FR = \{\text{F/R name, F/R shortened name, F/R primary key, F/R pointer, } n_2\} \cup P_s^{P+4}(AM) \cup P_s^{P+4+(7+P_1)}(AM) \cup P_s^{P+4+2(7+P_1)}(AM) \cup \ldots \cup P_s^{P+4+(n_2-1)(7+P_1)}(AM) \cup P_s^{P+4+(n_2)(7+P_1)} \{\text{End of Record}\}.$$

The corresponding structure for an attribute name for Group 2 is:

$$G2A = \{\text{attribute name, attribute shortened name, size descriptor, representation technique, coding function to the first pointed to synonym, } n_6\text{, pointer 1, pointer 2, ..., pointer } n_6\text{, uniqueness descriptor, password name, privileges name, } n_3\} \cup P_s^{9+n_6+P_1}(F/RM) \cup P_s^{9+n_6+P_1+(P+3)}(F/RM) \cup \ldots \cup P_s^{9+n_6+P_1+(n_3-1)(P+3)}(F/RM) \cup P_s^{9+n_6+P_1+(n_3)(P+3)} \{\text{End of Record}\}.$$

The pointers one through n_6 are the locations or pointers to the G2A structures of the synonyms of the attribute in question. The first pointer in each G2A, as shown in Fig. VII-7, form a circular linked list with each G2A data structure maintaining the coding function to the next synonym in the linked list. These two groups of structures describe a physical structure of the data dictionary and partial data directory implemented in the main memory of a sequential computer.

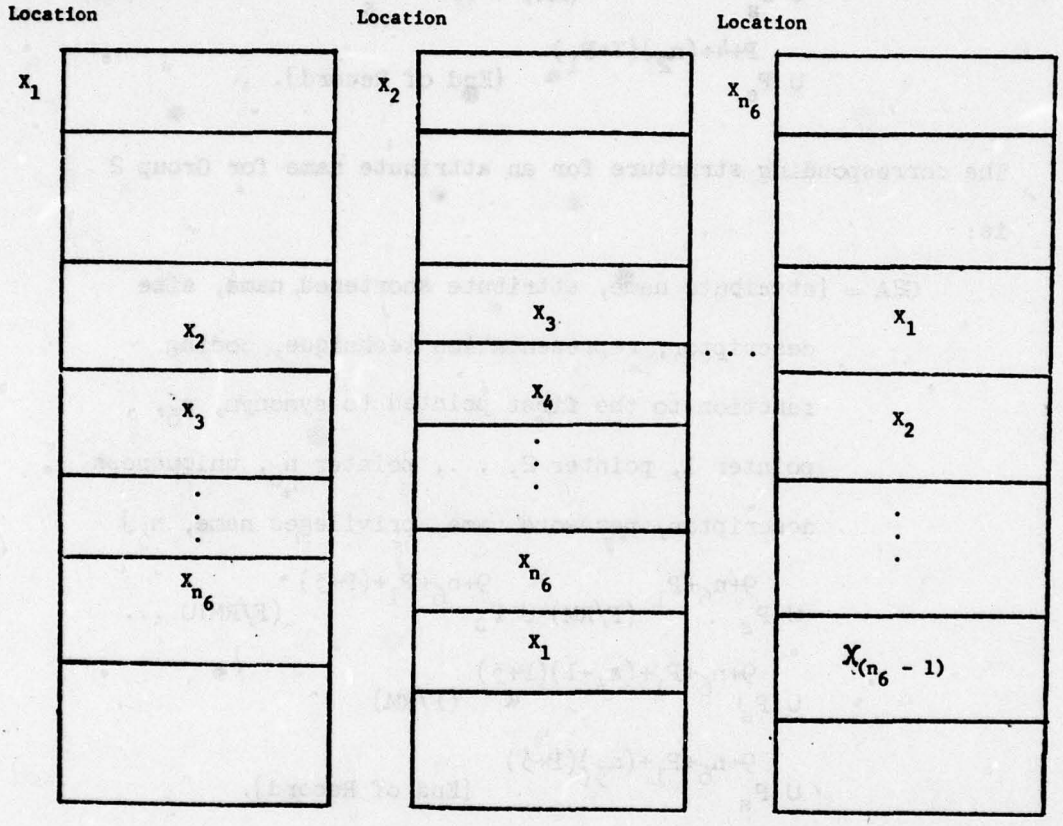


Figure VII-7. Group 2 Attribute Memory Arrangement for Type II Synonyms Stored in Locations X_1, X_2, ..., X_{n_6}.

DATA TRANSFER

The first portion of this section presents a general MIXAL code for transferring data from one portion of main memory to another. The data are assumed to be either G1FR, G1A, G2FR, or G2A data located in the data dictionary and partial directory. It is assumed the data are located by hashing on either an F/R name of P computer words long or an attribute name of P_1 computer words long. Once the data are chosen, the transfer code will move the data to the user's work area. The second portion of this section presents a development of a general expression for a timing equation for hashing and transferring data for variable key sizes.

General Transfer Code

The MIXAL code presented below will transfer n-w words of data from one portion of the computer's main memory to another portion starting at an address of BEGIN + w. w is the size, in computer words, of the hashing key that located the data to be transferred. It is assumed that the hashing has taken place and was successful. Therefore control was sent to location SUCCESS. This action will cause the following code to be executed:

Line No.	Location	Operation	Address	NTS	Explanation
01	SUCCESS	ENT3	=99=	1(1)	Sets r3 to 99
02		ENT2	BEGIN	1(1)	Sets r2 to the value of BEGIN

Line No.	Location	Operation	Address	NTS	Explanation
03		STA	w-1,2	1(2)	Stores the value of rA into the location BEGIN plus "w-1"
04		INC2	w	1(1)	Add the value w to the contents of r2
05		INC1	w	1(1)	Add the value w to the contents of r1
06	S.O.	COMP3	0,1	(n-w)(2)	Compare the contents of r3 with the contents of r1, i.e. searching for End of Record
07		JE	F.I.	(n-w)(1)	If they are equal then jump to F.I. (finished)
08		LDA	0,1	[n-(w+1)](2)	Load rA with the value in the address stored in r1, i.e. the next word in the record
09		STA	0,2	[n-(w+1)](2)	Store the contents of rA into the next word in the user's work area
10		INC1	1	[n-(w+1)](1)	Add the value 1 to the contents of r1
11		INC2	1	[n-(w+1)](1)	Add the value 1 to the contents of r2
12		JMP	S.O.	[n-(w+1)](1)	Jump to location S.O. (start over)
13	F.I.	LDA	0,1	(1)(2)	Load rA with the contents in the address stored in r1, i.e. 99, end of record marker

Line No.	Location	Operation	Address	NTS	Explanation
14		STA	0,2	(1)(2)	Store rA in the address stored in r2, i.e. 99, end of record marker, stored in user's work area.

END

$$\text{Total time} = \left(\sum_{\text{Statement 01}}^{14} \text{NTS} \right) U = (3 + 10(n-w))\, U.$$

In the above code, 99 signifies an end of record, BEGIN is the first location in the user's area in which the transfer data (G1FR, G1A, G2FR, or G2A) is to be stored contiguously, and "w-1" signifies the number of computer words that should be added to r2 to determine the location to store the lowest portion of the hashed key.

Time for Hashing and Transfer

An intuitive derivation for a general expression of the time for hashing on a key and then transferring the key's contiguous stored data to the user's work area is developed below. Consider the above code for a key size of one computer word, i.e. w = 1. This implies a hashing time of 34U and a transfer time of [(3 + 10(n-1)]U, where n is the number of words in the transfer data (i.e. the size of G1FR, G1A, G2FR, or G2A). The transfer code repeatedly operates on (n-1) words of the data to be transferred. The first word of the data is stored in the user's

area by the third statement in the above code, "STA "w-1", 2".

Consider the case when w = 2. This implies that the hashing time is [40 + (w-2)21]U, 40U and the transfer time is [3 + 10(n-2)]U. However, the summation of those two timing values is not the total time for hashing and transfer. The problem is that the highest portion of the key has not been stored in the user's work area. To rectify this situation, a statement STA 0,2 needs to be inserted between statements 13 and 14 of the hashing code for keys equal to two computer words (w = 2). Therefore, the total time would be 40U + [3 + 10(n-2)]U + 2U. The same procedure holds for the case when w = 3. The hashing time is [40 + (w-2)21]U or 61U plus the transfer time [3 + 10(n-2)]U plus 4U. The additional 4U is due to storing the highest and middle portions of the hashing key. The changes to the hashing code for keys three words long would occur between statements 16 and 17 and statements 19 and 20. The statement to be inserted between 16 and 17 would be STA 0,2 and inserted between 19 and 20, the statement would be STA 1,2. Therefore a general expression for the total time to hash and transfer data is:

$$\text{Total time} = \begin{cases} 34U + (3 + 10(n-1))U & \text{for } w = 1 \\ [40 + (w-2)21]U + [3 + 10(n-w)]U + 2(w-1)U & \text{for } w \geq 2 \end{cases}$$

where w is the key word size (in computer words),

n is the number of computer words to be transferred, and

U is a unit of time.

TIMING EQUATIONS FOR GENERIC JOBS 1 AND 2

Given the above MIXAL codes, the derivation of the timing equations for the first two generic jobs can be developed. These equations will be developed first. Then, timing equations will be developed for Jobs 3 and 4 using the timing results obtained for Job 2 and some additional MIXAL code.

Job 1

Given the F/R name, the DBMS is required to provide all of its occurrences. For Job 1 performed on a sequential computer there will be four different timing equations based on the following:

1) Hash on the F/R name whose size is one word long ($P = 1$) and the DBMS has Type I synonyms (T_{1SI1});
2) Hash on the F/R name whose size is ≥ 2 words long ($P \geq 2$) and the DBMS has Type I synonyms (T_{1SI2});
3) Hash on the F/R name whose size is one word long ($P = 1$) and the DBMS has Type II synonyms (T_{1SII1}); and
4) Hash on the F/R name whose size is ≥ 2 words long ($P \geq 2$) and the DBMS has Type II synonyms (T_{1SII2}).

Each timing equation represents the time to hash on the F/R name and transfer the proper data, i.e. either G1FR or G2FR depending on whether the DBMS has Type I or Type II synonyms respectively.

T_{1SI1}

The timing equation T_{1SI1} represents the time to hash on an F/R name one word long and to transfer G1FR to the user's area. Therefore, using the previously developed equations for (w = 1) yields the following:

Total time = 34U + (3 + 10(n-1))U.

Substituting $n = [P + 6 + 3n_1 + n_2(P_1+7)]$ into the above yields

$$T_{1SI1} = [97 + 30n_1 + 10n_2(P_1 + 7)]\ U.$$

T_{1SI2}

The timing equation T_{1SI2} represents the time to hash on an F/R name $\geq$ 2 words long and to transfer G1FR to the user's area. Using the previously developed equations for $w \geq 2$ yields the following:

Total time = [40 + (w-2)21]U + [3 + 10(n-w)]U + 2(w-1) U.

Substituting w = P and $n = (P + 6 + 3n_1 + n_2(P_1+7))$ into the above yields

$$T_{1SI2} = [59 + 23P + 30n_1 + 10(n_2(7+P_1))]\ U.$$

T_{1SII1}

The timing equation T_{1SII1} represents the time to hash on an F/R name one word long and to transfer G2FR to the user's area. Using the previously developed equations for w = 1 yields the following:

Total time = (34)U + (3 + 10(n-1)) U.

Substituting $n = [P + 5 + n_2(7+P_1)]$ into the above yields

$$T_{1SII1} = [42 + 10(n_2(7 + P_1))] U.$$

T_{1SII2}

The timing equation T_{1SII2} represents the time to hash on an F/R name $\geq$ 2 words long and to transfer G2FR to the user's area. Using the previously developed equations for $w \geq 2$ yields the following:

$$\text{Total time} = [40 + (w-2)21]U + [3 + 10(n-w)]U + 2(w-1) U.$$

Substituting $w = P$ and $n = [P + 5 + n_2(7+P_1)]$ into the above yields

$$T_{1SII2} = [49 + 23P + 10(n_2)(7 + P_1)] U.$$

Job 2

Given an F/R name and a number of its attribute names, the DBMS is required to provide a subset of the occurrences of the F/R. For Job 2, performed on a sequential computer, there will be four different timing equations based on the following:

1) Hash on the names of the n_5 attributes which are one word long ($P_1 = 1$), and the DBMS has Type I synonyms (T_{2SI1});
2) Hash on the names of the n_5 attributes whose size is $\geq$ 2 words long ($P_1 \geq 2$) and the DBMS has Type I synonyms (T_{2SI2});
3) Hash on the names of the n_5 attributes which are

one word long ($P_1 = \underline{1}$) and the DBMS has Type II synonyms (T_{2SII1}); and

4) Hash on the names of the n_5 attributes whose size is $\geq$ 2 words long ($P_1 \geq \underline{2}$) and the DBMS has Type II synonyms (T_{2SII2}).

To perform Job 2, when the DBMS has Type I synonyms, requires that Job 1 also be performed. This is necessary because some of the n_5 attribute's values may not be stored in the F/R in which the job was specified. Therefore, the DBMS would require the information F/RCM contained in G1FR to determine the F/Rs that are related to the F/R specified in the original job.

For Type I synonyms, each timing equation represents the time to hash on n_5 attribute names, to perform n_5 transfers with the proper data, i.e. G1A and to perform Job 1 for the given F/R name. For Type II synonyms, each timing equation represents the time to hash on n_5 attribute names and to perform n_5 transfers with the proper data, i.e. G2A.

T_{2SI1}

The timing equation T_{2SI1} represents the time to hash on n_5 attribute names which are one word long, perform n_5 transfers of the proper data (G1A) and to perform Job 1. Therefore, using the previously developed equations for (w = 1) yields the following:

Total time = $n_5(34)U + n_5\ [3 + 10(n-w)]U + T_{1SI1}$ (or T_{1SI2}).

Substituting $w = P_1$ and $n = [P_1 + 10 + n_6 + n_3(P+3)]$ in the above yields

$$T_{2SI1} = n_5[137 + 10(n_6 + (P+3)n_3)]U + [T_{1SI1} \text{ (or } T_{1SI2})].$$

T_{2SI2}

The timing equation T_{2SI2} represents the time to hash on n_5 attribute names which are ≥ 2 words long, perform n_5 transfers of the proper data (G1A) and to perform Job 1. Therefore, using the previously developed equations for $w \geq 2$ yields the following:

$$\text{Total time} = n_5[40 + (w-2)21]U + n_5[3 + 10(n-w)]U + 2n_5(w - 1)U.$$

Substituting $w = P_1$ and $n = [P_1 + 10 + n_6 + n_3(P+3)]$ in the above yields

$$T_{2SI2} = n_5[99 + 23P_1 + 10(n_6 + n_3(P+3))]U + [T_{1SI1} \text{ (or } T_{1SI2})].$$

T_{2SII1}

The timing equation T_{2SII1} represents the time to hash on n_5 attribute names which are one word long and perform n_5 transfers of the proper data (G2A). Therefore, using the previously developed equations for $(w = 1)$ yields the following:

$$\text{Total time} = n_5(34)U + n_5[3 + 10(n-w)]U.$$

Substituting $w = P_1$ and $n = [P_1 + n_6 + n_3(P+3) + 10]$ in the above yields

$$T_{2SII1} = n_5[137 + 10(n_6 + n_3(P+3))]U.$$

T_{2SII2}

The timing equation T_{2SII2} represents the time to hash on n_5 attribute names which are ≥ 2 words long and perform n_5 transfers

of the proper data (G2A). Therefore, using the previously developed equations for $w \geq 2$ yields the following:

$$\text{Total time} = n_5[40 + (w-2)21]U + n_5[3 + 10(n-w)]U + (n_5)\,2(w-1)\,U.$$

Substituting $w = P_1$ and $n = [P_1 + n_6 + n_3(P+3) + 10]$ in the above yields

$$T_{2SII2} = n_5[99 + 23P_1 + 10(n_6 + n_3(P+3))]\,U.$$

TIMING EQUATIONS FOR GENERIC JOB 3

Job 3: Given an attribute's occurrence(s) of Type II synonyms, the DBMS is required to modify (e.g. delete, change value, etc.) the attribute and its synonym's occurrences in all their associated F/Rs. The derivation of the timing equations for Job 3 is different than that used for Jobs 1 and 2. Consider the following two situations for Job 3. The first situation is that, say Job 1, has been performed and it is then decided that an attribute in the F/R in question needs to be modified. The procedure would be to hash on the attribute's name and then transfer its G2A data to the user's area. Then the DBMS must locate through the synonym pointers of the G2A all the attribute's synonym's G2A data and transfer those data to the user's area.

The second situation that can occur for Job 3 is that the G2A data, of the attribute in question, is currently in the user's area. This could have occurred through a previous Job 2 run. The procedure for Job 3 would involve locating, through the synonym pointers of the G2A, all the attribute's synonym's G2A data and transfer those data to the user's area.

T_{3SII1}

The timing equation T_{3SII1} is for the first situation and represents the time to perform Job 3 on a sequential computer with Type II synonyms. It represents the 1st equation for Job 3 and includes the time to:

1) hash on an attribute's name one word long;
2) transfer its G2A to the user's area; and
3) search and transfer the attribute's synonym's G2A data.

The timing equation for the first two parts is simply T_{2SII1} solved for $n_5 = 1$, i.e.

$$T_{2SII1} = (34)U + [3 + 10(n_6 + n_3(P+3) + 10)]\ U,$$

$$T_{2SII1} = [137 + 10(n_6 + n_3(P+3))]\ U.$$

Once the G2A data are in the user's area, the third part of T_{3SII1} can be accomplished by the following MIXAL code which will provide the user's area with the attribute's synonym's G2A data. This code would be executed directly following the general transfer code associated with T_{2SII1}.

Line No.	Location	Operation	Address	NTS	Explanation
02		LD5	"BEGIN+P_1+5"	(2)(1)	Loads the contents of r5 with the value in location "BEGIN+P_1+5," i.e. the number of pointers (n_6)

Line No.	Location	Operation	Address	NTS	Explanation
03		LD6	"BEGIN+P_1+6"	(2)(1)	Loads the contents of r6 with the value in location "BEGIN+P_1+6", i.e. the first address (or pointer) to a synonym's G2A data
04		ENT4	"BEGIN+P_1+6"	(1)(1)	Sets r4 to the value "BEGIN+P_1+6"
05		ENTA	0	(1)(1)	Sets rA to zero
06		STA	L	(2)(1)	Stores rA in address L (i.e. stores a zero)
07	S.O.	COMP5	L	(2)(n_6+1)	Compare r5 with the contents in L (i.e. zero)
08		JE	DONE	(1)(n_6+1)	If r5 is equal to zero, jump to DONE
09		INC2	1	(1)(n_6)	Add 1 to r2 which advances the location in the user's work area
10	N.W.	COMP3	0,6	(2)(n)	Compare the contents of the location stored in r6 with the contents of r3 (i.e. 99, end of record marker)
11		JE	N.S.	(1)(n)	If equal, jump to N.S. (next synonym)
12		LDA	0,6	(2)(n)	Load rA with contents of r6 (a word in a G2A)

Line No.	Location	Operation	Address	NTS	Explanation
13		STA	0,2	(2)(n)	Store the contents of rA into r2 (stores a word in a G2A record into the user's work area)
14		INC2	1	(1)(n)	Add 1 to r2
15		INC6	1	(1)(n)	Add 1 to r6
16		JMP	N.W.	(1)(n)	Jump to N.W. (next word)
17	N.S.	LDA	0,6	$(2)(n_6)$	Load rA with the contents of the address in r6 (i.e. 99)
18		STA	0,2	$(2)(n_6)$	Store the contents of rA (99) in the address contained in r2 (i.e. the next word in the user's work area)
19		DEC5	1	$(1)(n_6)$	Subtract 1 from r5
20		INC4	1	$(1)(n_6)$	Add 1 to r4
21		LD6	0,4	$(2)(n_6)$	Loads r6 with the value in the location stored in r4 (i.e. the address of the next G2A record)
22		JMP	S.O.	$(1)(n_6)$	Jump to S.O. (start over)
23	DONE	END			

$$\text{Total time} = \left(\sum_{\text{Statement 02}}^{23} \text{NTS} \right) U = 11U + 13(n_6)U + 10(n_6)(n)U$$

In the above code S.O. represents Start Over, N.S. represents Next Synonym, N.W. represents Next Word, and "BEGIN+P_1+5" and "BEGIN+P_1+6" are the locations containing n_6 and the first pointer, respectively. The total time for the above code, by replacing n with $[P_1 + n_6 + n_3(P+3) + 10]$ for G2A, is

$$\text{Total time} = (11)U + 13(n_6)U + 10(n_6)[P_1 + n_6 + n_3(P+3) + 10]\,U.$$

Therefore, summing the times for the three parts of T_{3SII1} yields:

$$T_{3SII1} = T_{2SII1} \text{ (with } n_5 = 1) + 11U + 113(n_6)U + 10n_6[P_1 + n_6 + n_3(P+3)]U.$$

T_{3SII2}

The timing equation T_{3SII2} is for the first situation and represents the time to perform Job 3 on a sequential computer with Type II synonyms. It represents the 2nd equation for Job 3 and includes the time to:

1) hash on an attribute's name $\geq$ 2 words long;
2) transfer its G2A data; and
3) search and transfer the attribute's synonym's G2A data.

The timing equation for the first two parts is simply T_{2SII2} solved for $n_5 = 1$, i.e.

$$T_{2SII2} = [40 + (P_1-2)21]U + [10(P_1 + n_6 + n_3(P+3) + 10) - 8P_1 + 1]\,U,$$

$$T_{2SII2} = [99 + 23P_1 + 10(n_6 + n_3(P+3))]U.$$

Once the G2A data are in the user's area, then the same MIXAL code applied to T_{3SII1} would be executed to obtain the third part of T_{3SII2}. Therefore, summing the times for the 3 parts of T_{3SII2} yields

$$T_{3SII2} = T_{2SII2} \text{ (with } n_5 = 1) + (11)U + 13(n_6)U + 10(n_6)$$

$$[P_1 + n_6 + n_3(P + 3) + 10]U.$$

T_{3SII3}

The timing equation T_{3SII3} is for the second situation and represents the time to perform Job 3 on a sequential computer with Type II synonyms. It represents the 3rd equation for Job 3 and includes the time to search the G2A data in the user's area and transfer the attribute's synonym's G2A data in the user's area. This timing equation is obtained by realizing that to accomplish this searching and transfer of data the MIXAL code for situation one must be executed. It is essentially equivalent to the third part of the timing equations for the first situation. The only difference is that the above code would be prefaced by the following two statements, because T_{2SII1} or T_{2SII2} are not assumed to be executed before the above MIXAL code.

Line No.	Location	Operation	Address	NTS	Explanation
00		ENT2	Z	1	Set r2 equal to Z, the starting location where the G2A data are to be stored
01		ENT3	=99=	1	Set r2 equal to 99, signifying an end of record

$$\text{Total time} = \left(\sum_{\text{Statement 00}}^{01} \text{NTS} \right) U = 2U.$$

Therefore, summing the times for the total MIXAL code yields

$$\text{Total time} = (11)U + 13(n_6)U + 10(n_6)(n) + 2U.$$

Substituting $n = [P_1 + n_6 + n_3(P+3) + 10]$ in the above yields

$$T_{3SII3} = (13)U + 113(n_6)U + 10(n_6)[P_1 + n_6 + n_3(P+3)]\ U.$$

TIMING EQUATIONS FOR GENERIC JOB 4

Given an attribute's occurrence(s) of Type I synonyms, the DBMS is required to modify (e.g. delete, change value, etc.) the attribute's stored synonym occurrence(s) in its associated F/R. To derive the timing equations for Job 4 requires an approach similar to Job 3. Consider the following two situations for Job 4. The first situation is that, say Job 1, has been performed and it is then decided that an attribute in the F/R in question needs to be modified. The procedure would be to hash on the attribute's name and then transfer its G1A data to the user's

area. Then the DBMS must locate through the first synonym pointer the GlA data for the stored synonym and transfer these data to the user's area.

The second situation that can occur for Job 4 is that the GlA data, of the attribute in question, is currently in the user's area. This could have occurred through a previous Job 2 run. The procedure for Job 4 would involve locating, through the first synonym pointer of the GlA, the GlA data for the stored synonym and transferring this data to the user's area.

T_{4SI1}

The timing equation T_{4SI1} is for the first situation and represents the time to perform Job 4 on a sequential computer with Type I synonyms. It represents the 1-st equation for Job 4 and includes the time to:

1) hash on an attribute's name one word long;
2) transfer its GlA data to the user's area; and
3) search and transfer the stored synonym's GlA data.

The timing equation for the first two parts is simply T_{2SI1} solved for $n_5 = 1$ and minus the time to perform Job 1, i.e.

$$T_{2SI1} = [137 + 10(n_6 + n_3(P+3))]\ U.$$

Once the GlA data is in the user's area, the third part of T_{4SI1} can be accomplished by the following MIXAL code which will provide in the user's area the attribute's stored synonym GlA data. This code would be executed directly following the general transfer code associated with T_{2SI1}.

Line No.	Location	Operation	Address	NTS	Explanation
02		LD5	"BEGIN+P_1+5"	(2)	Loads the contents of r5 with the value in location "BEGIN+P_1+5", i.e. the address of the first word for the stored synonym's data (G1A)
03		LD6	"BEGIN+P_1+6"	(2)	Loads the contents of r6 with the value in location "BEGIN+P_1+6", i.e. the address of the first word for the 2nd synonym's data (G1A)
04		ENTA	0	(1)	Sets rA to zero
05		STA	L	(2)	Stores the contents of rA into location L, i.e. stores a zero in L
06		COMP5	L	(2)	Compares the contents of r5 with the contents of L to check the address which should be equal to zero
07		JE	N.S.	(1)	Jump to N.S. (next synonym) if they are equal. This signifies that it has the stored value.
08		COMP6	L	(2)	Compares the contents of r6 with the contents of L--to check if the address is equal to zero

Line No.	Location	Operation	Address	NTS	Explanation
09		JE	IIOS	(1)	Jump to IIOS (it's its own synonym) if they are equal--signifying that it has no synonyms
10		INC2	1	(1)	Add one to the contents of r1
11	N.W.	COMP3	0,5	2(n)	Compare contents of r3 with the contents of the word whose address is stored in r5 (checks for end of record)
12		JE	DONE	1(n)	If they are equal, jump to DONE
13		LDA	0,5	2(n)	Load rA with the contents of the word whose address is in r5
14		STA	0,2	2(n)	Store the contents of rA into the word whose address is in r2
15		INC2	1	1(n)	Add one to the contents of r2
16		INC5	1	1(n)	Add one to the contents of r5
17		JMP	N.W.	1(n)	Jump to N.W. (next word)
		N.S.	END		
		IIOS	END		
		DONE	END		

$$\text{Total time} = \left(\sum_{\text{Statement 02}}^{17} \text{NTS} \right) U = (14)U + 10(n)\ U.$$

The total time for the above code by replacing n with $[P_1 + n_6 + n_3(P+3) + 10]$ in the above yields

$$\text{Total time} = (14)U + 10[P_1 + n_6 + n_3(P+3) + 10]\ U.$$

Therefore, summing the times for the three parts of T_{4SI1} yields

$$T_{4SI1} = (251)U + 10[P_1 + 2n_6 + 2n_3(P+3)]\ U.$$

T_{4SI2}

The timing equation T_{4SI2} is for the first situation and represents the time to perform Job 4 on a sequential computer with Type I synonyms. It represents the 2nd equation for Job 4 and includes the time to:

1) hash on an attribute's name $\geq$ 2 words long;
2) transfer its GIA data to the user's area; and
3) search and transfer for the stored synonym's GIA data.

The timing equation for the first two parts is simply T_{2SI2} solved for $n_5 = 1$ and minus the time to perform Job 1, i.e.

$$T_{2SI2} = [99 + 23P_1 + 10(n_6 + n_3(P+3))]\ U.$$

Once the GIA data is in the user's area, then the same MIXAL code applied to T_{4SI1} would be executed to obtain the third part of T_{4SI2}. Therefore, summing the times for the 3 parts of T_{4SI2} yields

$$T_{4SI2} = (213 + 3P_1)U + 20[n_6 + n_3(P+3)]\ U.$$

T_{4SI3}

The timing equation T_{4SI3} is for the second situation and represents the time to perform Job 4 on a sequential computer with Type I synonyms. It represents the 3-rd equation for Job 4 and includes the time to search GlA data in the user's area and transfer the attribute's stored synonym's GlA data to the user's area. This timing equation is obtained by realizing that to accomplish this searching and transfer of data the MIXAL code for situation one must be executed. It is essentially equivalent to the third part of the timing equations for the first situation. The only difference is that the above code would be prefaced by the following two statements because T_{2SI1} or T_{2SI2} are not assumed to be executed before the above MIXAL code.

Line No.	Location	Operation	Address	NTS	Explanation
00		ENT2	Z-1	(1)	Sets r2 with a location one less than where the data is to begin being stored
01		ENT3	=99=	(1)	Sets r3 with a value 99, i.e. end of record marker

$$\text{Total time} = \left(\sum_{\text{Statement 00}}^{01} \text{NTS}\right)U = 2U.$$

Therefore, summing the times for the total MIXAL code yields

$$\text{Total time} = (14)U + 10(n)U + 2U.$$

Substituting $n = [P_1 + n_6 + n_3(P+3) + 10]$ in the above yields

$$T_{4SI3} = [116 + 10(P_1 + n_6 + n_3(P+3)] \text{ U.}$$

EVALUATION RESULTS

INTRODUCTION

Figures VII-8 through VII-13 represent the results of evaluating the DDP by comparing its timing equations with those timing equations developed for a sequential computer, with a unit of time U chosen as one microsecond. The timings for the DDP were based on the STARAN AP and are shown as the micro-function timings in the first portion of this chapter. It is possible, by changing these timings, to evaluate different combinations of conventional sequential computers with different hardware making up the components of a DDP.

The timing equations plotted in the following figures are for a time increment to perform four generic jobs submitted to a DBMS. The timing increment for the DDP is from when the sequential computer's CPU notifies the DDP control of a job to be executed until the DDP supplies the TUMA with its results (See Fig. VI-1). The timing increment for the sequential computer is that time to locate data in the data dictionary/directory and transfer it to the user's memory area. The data dictionary/directory data are assumed to reside in the sequential computer's main memory.

The curves in the figures represent the maximum and minimum times for the sequential computer and the DDP. The maximum and

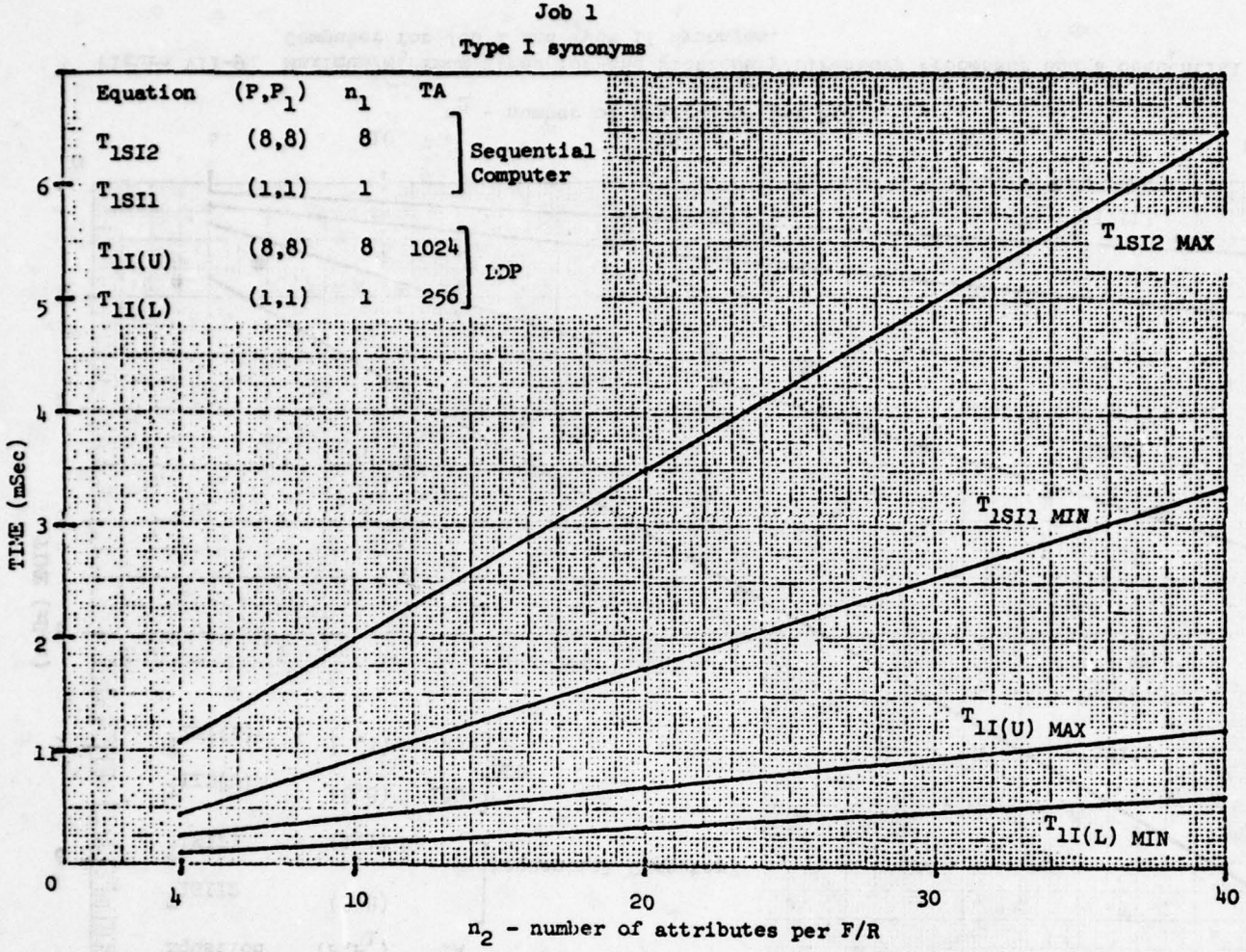


Figure VII-8. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 1 and Type I Synonyms

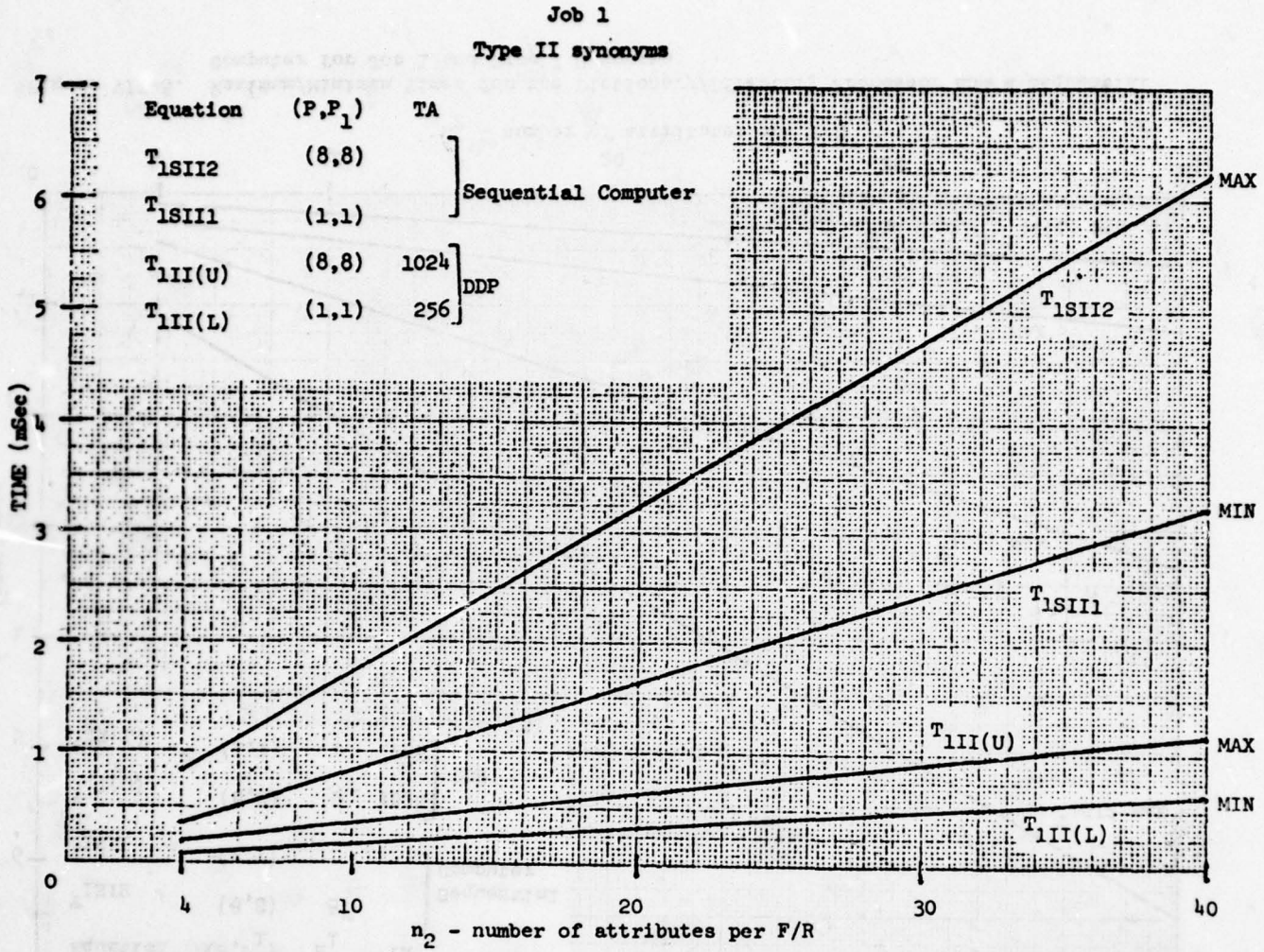


Figure VII-9. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 1 and Type II Synonyms.

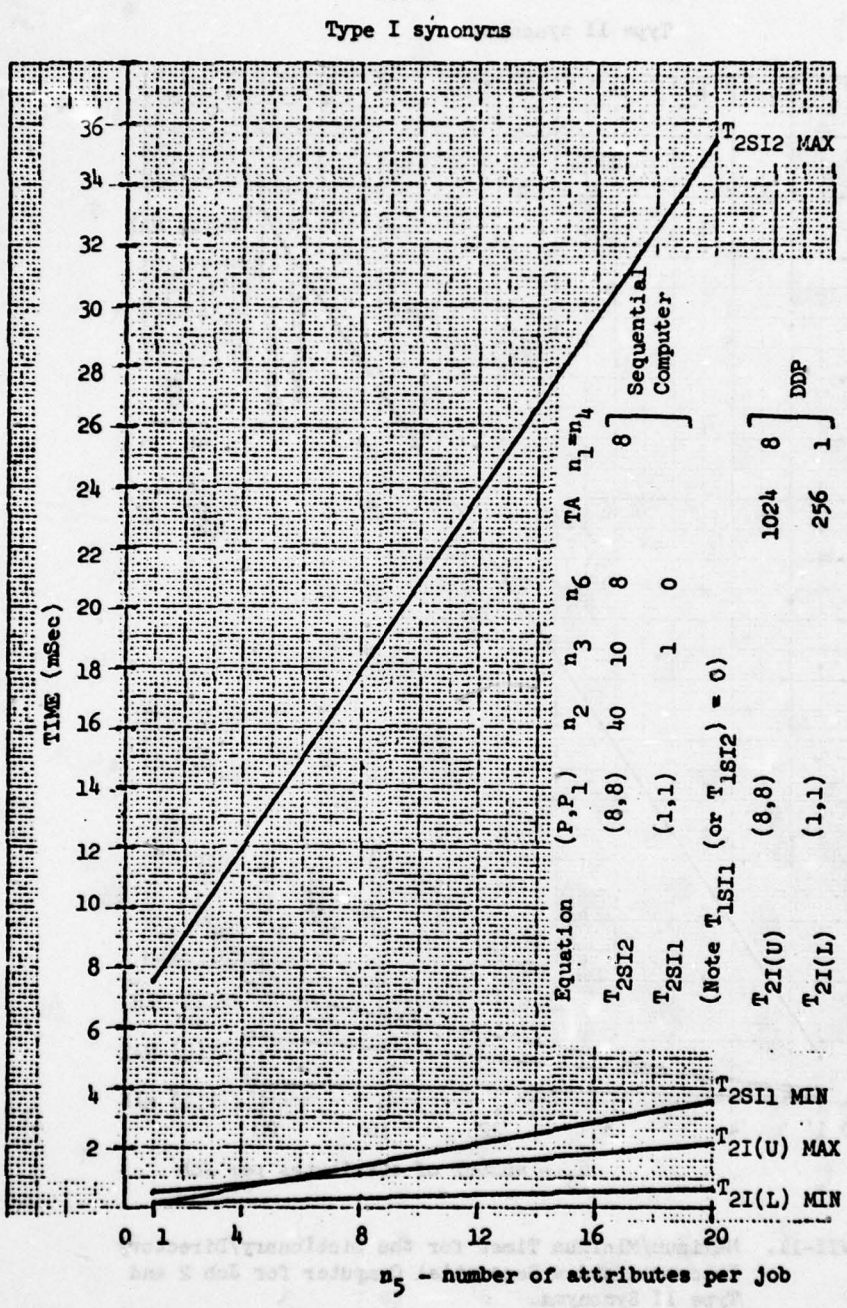


Figure VII-10. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 2 and Type I Synonyms.

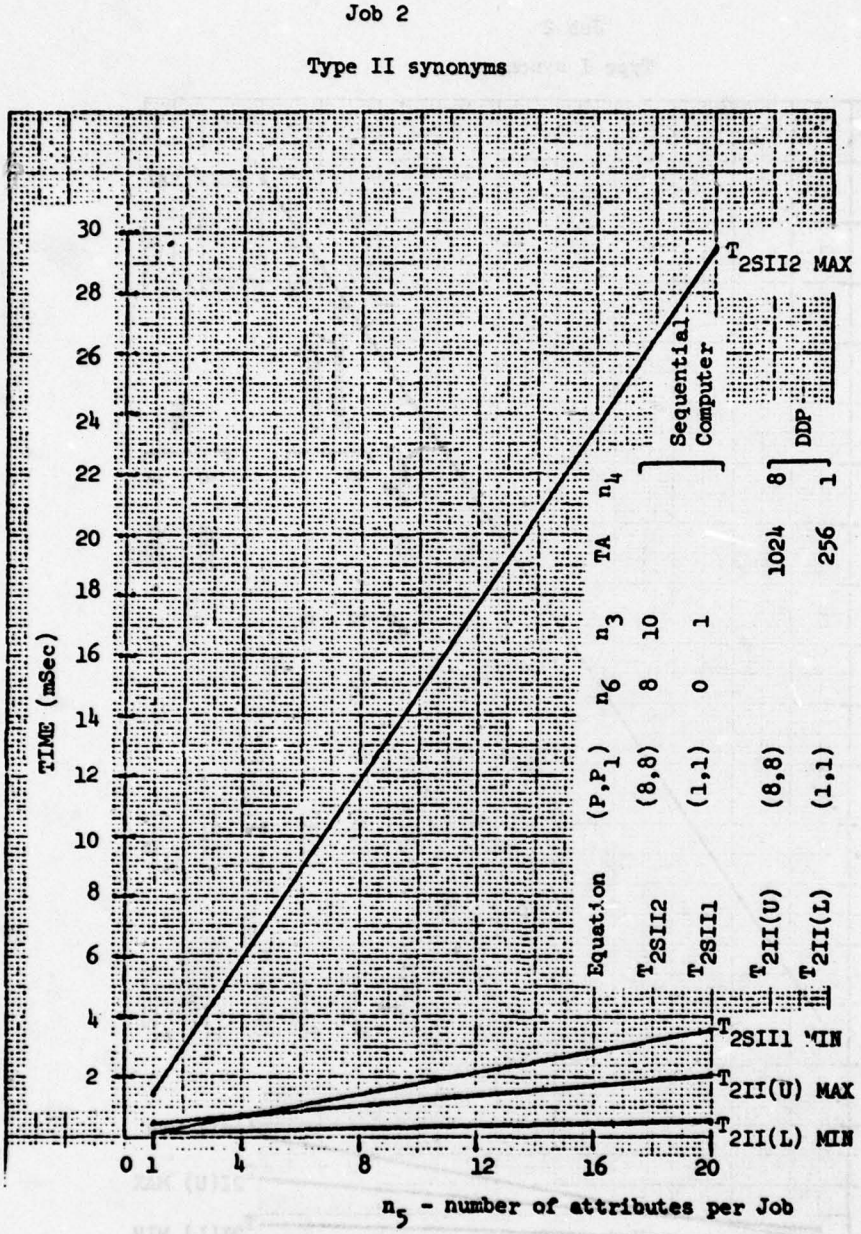


Figure VII-11. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 2 and Type II Synonyms.

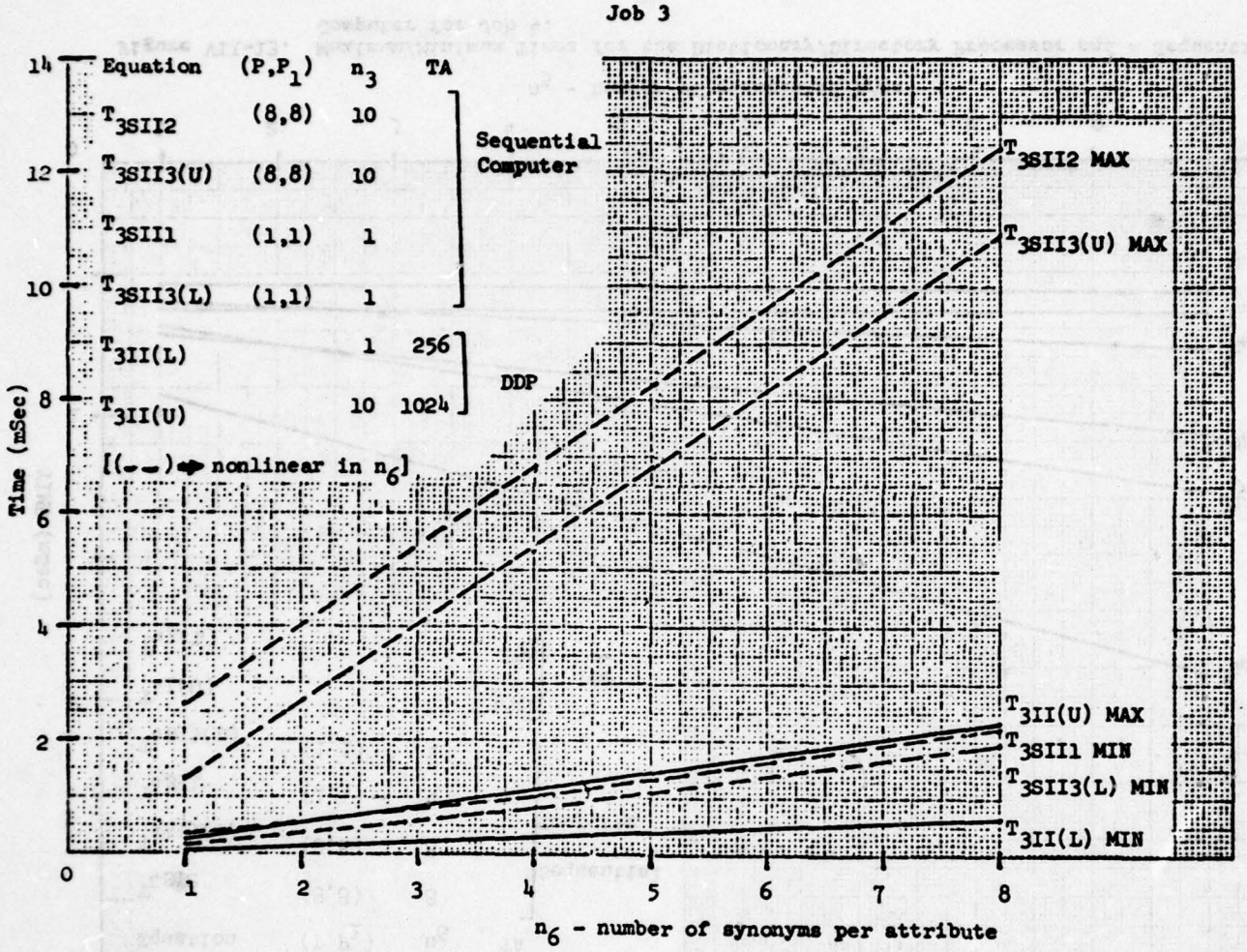


Figure VII-12. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 3.

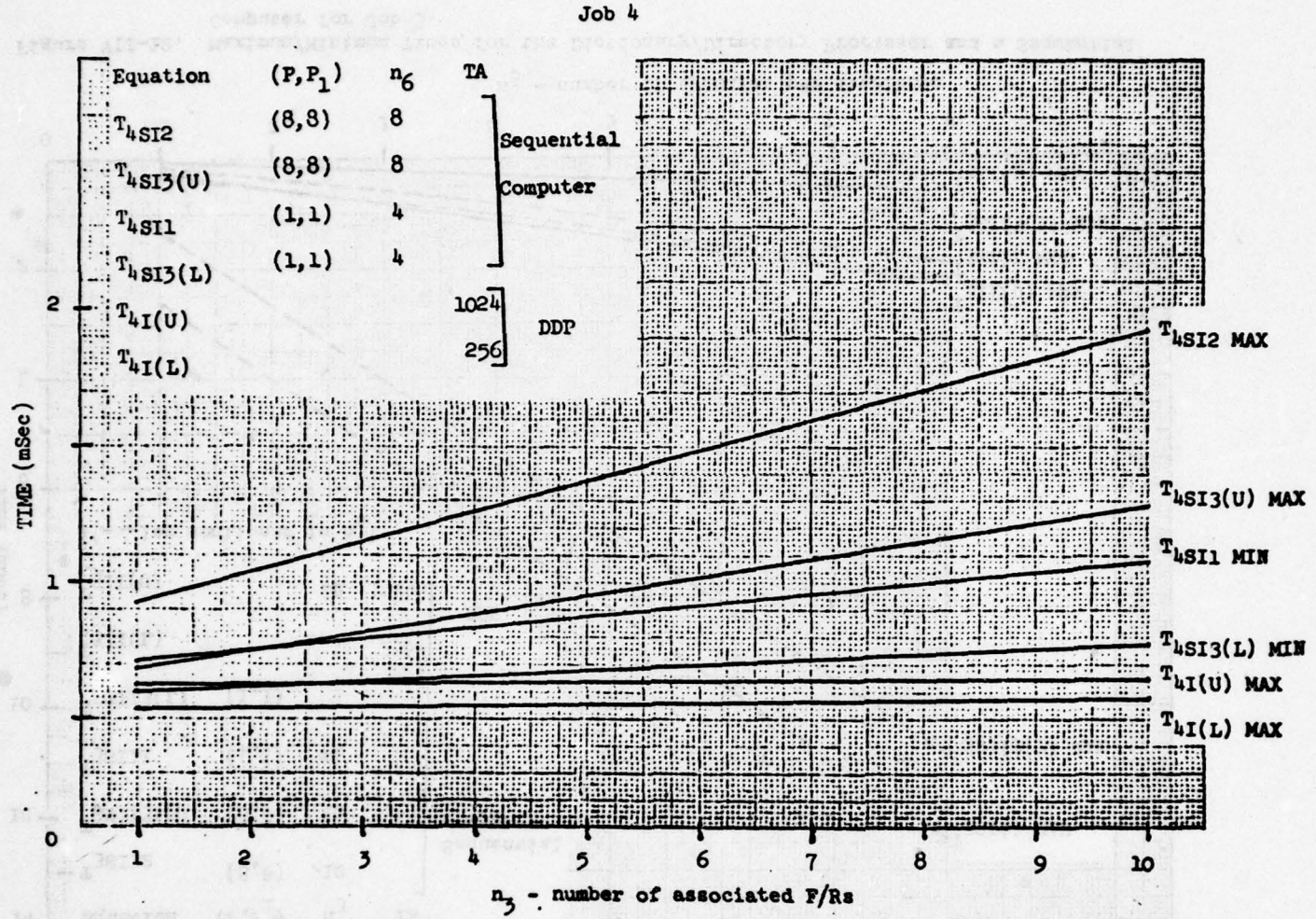


Figure VII-13. Maximum/Minimum Times for the Dictionary/Directory Processor and a Sequential Computer for Job 4.

minimum times are achieved by solving the equations for the curves with feasibly high and low values of their parameters. The maximum feasible values of some of these parameters were chosen because of the physical constraints of the STARAN AP (i.e. P, P_1, and TA). These curves provide a bounding in time per job of the DDP and the sequential computer and a comparison in time between the sequential computer and the DDP.

Jobs 1 and 2 are contained in two figures. One figure is for Type I synonyms and the other for Type II synonyms. Jobs 3 and 4 are for modifying synonyms Type I and II. Note that synonyms exist in a data base if two or more attributes have different names and descriptors or different names for the same entity in the real world and that Type I and II synonyms are methods of handling synonyms in a computer, where:

1) Type I. Store the occurrences of the synonyms in only one description and develop additional functions to convert the different occurrences from one description to another.
2) Type II. Store the occurrences of the synonyms in their different descriptions, e.g. coding, size, etc.

There are three basic assumptions that have been made in the development of these figures. They are:

1) all the jobs submitted to the DBMS are correct, i.e. they will not be aborted;
2) all equal to searches (ETS) performed on parameter values are found; and

3) the hashing algorithm employed for the sequential computer implementation does not consider collisions.

Job 1

Job 1 requires the DBMS to provide all the occurrences of an F/R when provided with only the F/R name. The results of analyzing the two implementations on a sequential computer and the DDP are shown in Fig. VII-8 and VII-9. The lower two curves of Fig. VII-8 ($T_{1I(L)}$, $T_{1I(U)}$) and Fig. VII-9 ($T_{1II(L)}$ and $T_{1II(U)}$) show the minimum and maximum times for the DDP implementation. The upper curves (T_{1SI1}, T_{1SI2}) and (T_{1SII1}, T_{1SII2}) show the minimum and maximum times for the sequential computer implementation. The ordinate is in milliseconds and the abscissa represents values of n_2, the number of attributes associated with an F/R. The other parameters are:

1) P, the number of computer words per F/R name;
2) P_1, the number of computer words per attribute name;
3) TA, the total number of attributes in the data base; and
4) n_1, the number of F/Rs related to a given F/R and supported by the DBMS.

Comparing Figs. VII-8 and VII-9 reveals that the type of synonym only slightly affects the timings. The DDP is faster than the sequential computer and as n_2 increases, the ratio of the sequential computer time to the DDP time also increases.

Job 2

Job 2 requires the DBMS to provide a subset of the occurrences of an F/R when given the F/R's name and a number of its attribute names. The lower two curves in Fig. VII-10 ($T_{2I(L)}$, $T_{2I(U)}$) and Fig. VII-11 ($T_{2II(L)}$, $T_{2II(U)}$) show the minimum and maximum times for the DDP implementation. The upper two curves (T_{2SI1}, T_{2SI2}) and (T_{2SII1}, T_{2SII2}) show the minimum and maximum times for the sequential computer implementation. The ordinate is in milliseconds and the abscissa represents values of n_5, the number of attributes specified in the job. The other parameters are:

1) P, the number of computer words per F/R name;
2) P_1, the number of computer words per attribute name;
3) TA, the total number of attributes in the data base;
4) n_3, the number of associated F/Rs of a given attribute;
5) n_4, the number of associated F/Rs of a given number (n_5) of attributes;
6) n_6, the number of synonyms of a given attribute; and
7) n_1, the number of F/Rs related to a given F/R and supported by the DBMS.

Comparing Figs. VII-10 and VII-11 reveals that Type I synonyms increase the upper bound for the sequential computer implementation. The DDP is always faster than the sequential computer when n_5 is greater than six and their lower bounds are approximately equal where n_5 is equal to one. The ratio of the sequential computer time to the DDP time increases as n_5 increases.

Job 3

Job 3 requires the DBMS to modify an attribute's occurrence in a data base which has Type II synonyms. The DBMS is required to modify all (F/R)'s in which the attribute and the attribute's synonyms are contained. The curves labeled $T_{3II(L)}$ and $T_{3II(U)}$ in Fig. VII-12 are the lower and upper boundary curves for the DDP. The T_{3SII1} and T_{3SII2} are the lower and upper boundary curves for the sequential computer implementation assuming Job 1 had been previously run. The $T_{3SII3(L)}$ and $T_{3SII3(U)}$ are the lower and upper boundary curves for the sequential computer implementation assuming Job 2 had been previously run. The ordinate is in milliseconds and the abscissa represents values of n_6, the number of synonyms of a given attribute. The other parameters are:

1) P, the number of computer words per F/R name;
2) P_1, the number of computer words per attribute name;
3) n_3, the number of associated F/Rs of a given number (n_5) of attributes; and
4) TA, the total number of attributes in the data base.

Comparing the curves in Fig. VII-12 reveals that the lower bound of the DDP is always faster than a sequential computer implementation and the ratio of the sequential computer time to the DDP time increases as n_6 increases. The curves also reveal that Job 3 performed after Job 2 is faster than if performed after Job 1.

Job 4

Job 4 requires the DBMS to modify an attribute's occurrence in a data base which has Type I synonyms. The DBMS is required to modify the F/R in which the stored attribute's synonym is contained. The curves labeled $T_{4I(L)}$ and $T_{4I(U)}$ in Fig. VII-13 are the lower and upper boundary curves for the DDP. The T_{4SI1} and T_{4SI2} are the lower and upper boundary curves for the sequential computer implementation assuming Job 1 had been previously run. The $T_{4SI3(L)}$ and $T_{4SI3(U)}$ are the lower and upper boundary curves for the sequential computer implementation assuming Job 2 had been previously run. The ordinate is in milliseconds and the abscissa represents values of n_3, the number of associated F/Rs of a given attribute. The other parameters are:

1) P, the number of computer words per F/R name;
2) P_1, the number of computer words per attribute name;
3) n_6, the number of synonyms of a given attribute; and
4) TA, the total number of attributes in the data base.

Comparing the curves in Fig. VII-13 reveals that the lower bound of the DDP is always faster than a sequential computer implementation and the ratio of the sequential computer time to the DDP time increases as n_3 increases. The curves also reveal that Job 4 performed after Job 2 is in general faster than if performed after Job 1.

SUMMARY

This concludes the results of evaluating the DDP. This chapter has provided a description of the development and utilization of timing equations. These equations modeled both the DDP and a sequential computer's implementation of the functions performed by the DDP. The results of evaluating the DDP are presented; in general, the DDP is faster than the sequential computer implementation. The primary contribution provided in this chapter was the mathematical equations developed such that any conventional sequential computer and different DDP hardware components may be evaluated.

Chapter VIII

RESULTS, CONCLUSIONS, AND FUTURE RESEARCH

INTRODUCTION

The first three chapters contain an introduction, a review of the pertinent literature, the problem definition, and a methodology for approaching the problem. Chapters IV through VII contain a mathematical base for data base management which includes four levels of data, a proposed hardware design, and an evaluation of the design. The remainder of this chapter concentrates on the contents of these four chapters.

Chapter IV contained a description of a mathematical base for data base management. This base was composed of set theory and an extension of set theory called Data Processing (DP) sets. The relationships between sets and DP sets were also provided. The most interesting level of the mathematical base presented dealt with characteristic functions, characteristic sets, and the properties of each. This level provided a convenient method of implementing this mathematical base in computer hardware.

A language capability for modeling the levels and functions of Data Base Management is provided by sets and DP sets. Chapter V contains a presentation of this modeling pertaining to the following four levels of data:

1) the user computer interface (Reserved Word);

2) the attribute and file or relationship (F/R)

names (Data Name);

3) the modifiers of the attribute and F/R names (Data Descriptors); and

4) the occurrences of the attributes and F/Rs (Data Occurrence).

DBM functions are not divided into any specific levels. They are dynamic and operate on the above four levels of data to perform the jobs requested by the user of a DBM System (DBMS).

The sixth chapter contained a comparison of those levels of DBM described in Chapter V with special hardware already developed and described in the literature. It was determined that the Data Dictionary and part of the Data Directory (or parts of the Data Name and Data Descriptor levels) had not previously been totally addressed by the hardware community. The rest of the chapter describes a hardware implementation of a data dictionary and partial Directory of data Dictionary/Directory Processor (DDP). The description is presented in four steps. The first step interfaced the hardware with a sequential computer. Next, a design of the hardware at a more detailed level considering its control, data storage, and data transfer was given. The hardware at the logic gate and flip flop levels was provided in the third step. The final step demonstrated how the DDP would perform its DBM functions.

The seventh chapter contained an evaluation of the hardware design of the DDP. The mathematical equations developed for this evaluation can be used to evaluate any conventional sequential computer as well as various DDP hardware components. The

evaluation was provided through five steps which concluded by comparing the times required by the DDP and a sequential computer to perform four basic, yet all encompassing jobs that could be submitted to a DBMS. This chapter begins by providing some basic results and conclusions that can be drawn from Chapters IV through VII. This is followed by a discussion of three areas where future research should be performed.

RESULTS AND CONCLUSIONS

To complete the documentation of the results obtained from this research, two areas need to be addressed. The first is concerned with tying together the work described in Chapters IV and V with that described in Chapters VI and VII. The second is concerned with providing some numerical values resultant from the evaluation contained in Chapter VII.

There are important relationships tying together the contents of Chapters IV and V with that of VI and VII. In a previous chapter it was shown that the mathematical base developed (see Chapter IV) can be used (see Chapter V) to model DBM from that level seen and utilized by a user down to the bit level of a digital computer. Chapters VI and VII provided a description and an evaluation of a proposed hardware implementation of some of those DBM functions modeled in Chapter V. These functions operated on those levels of data pertaining to the data dictionary and a partial data directory (DDP). This hardware implementation of a DDP was developed by implementing portions of the mathematical base described in Chapter IV. This can be seen by comparing the

properties of Data Processing Characteristic sets (DPCSs) to those functions discussed in Chapters VI and VII and referred to in Table VII-1. To be specific, a single equal to search in an AM (see Table VII-1, Function I, Macro-function SETS) is an implementation in hardware of property XXIII for DPCSs. The macro-function SYNO and RELA of functions I and IV respectively in Table VII-1, is an implementation of the DPCS property XXIV. Finally, the ATFR macro-function of function IV in Table VII-1 is an implementation of DPCS XXI. Therefore, sets and DP sets not only provide a language for modeling DBM from the user's level down to the bit level but also provides a convenient method for implementing some of the DBM functions in hardware.

The DDP was evaluated (see Chapter VII) by comparing the time for the DDP and a sequential computer to execute the same jobs. These jobs were created to encompass the most basic or generic jobs a user would submit to a DBM system. The jobs are defined below, followed by the definitions of those parameters that influence the times for the DDP and the sequential computer to perform these generic jobs. Finally, Tables VIII-1 through VIII-3 will be discussed. They contain the numerical values of the minimum and maximum times for the DDP and the sequential computer to perform the generic jobs for a range of values of the above-mentioned parameters.

The four generic jobs that are created are:

1) Job 1 - to provide all the occurrences of a File/Relationship (F/R) given only the F/R name;

2) Job 2 - to provide a subset of the occurrences of an F/R given the F/R name and some operation on a number (n_5) of its attributes;

3) Job 3 - to modify all the occurrences of an attribute in a data base with Type II synonyms;* and

4) Job 4 - to modify all the occurrences of an attribute in a data base with Type I synonyms.*

The parameters that influence the times to perform the above generic jobs are:

1) P, the number of computer words per F/R name;

2) P_1, the number of computer words per attribute name;

3) n_1, the number of F/Rs related to a given F/R and supported by the DBMS;

4) n_2, the number of attributes in a given F/R;

5) n_3, the number of associated F/Rs of a given attribute;

6) n_4, the number of associated F/Rs of a given number (n_5) of attributes;

7) n_5, the number of attributes specified in Job 2;

8) n_6, the number of synonyms of a given attribute; and

9) TA, the total number of attributes in the data base.

Table VIII-1 contains the minimum and maximum times in milliseconds for Job 1. The three double columns of data are

---

*Note: Type I and II synonyms are methods of handling synonyms in a computer where:
1) Type I - The occurrences of the synonyms are stored in only one description and additional functions are developed to convert the different occurrences from one description to another; and
2) Type II - The occurrences of the synonyms are stores in their different descriptions.

the times for an F/R which has 10, 20, and 40 (n_2) attributes. The first two rows of data provide the times for the sequential computer with Type I and Type II synonyms, respectively. The following two rows of data provide the times for the DDP with Type I and Type II synonyms, respectively. The parameters and their values for the minimum columns of data are $P = P_1 = n_1 = 1$ and TA = 256. The parameters and their values for the maximum columns of data are $P = P_1 = n_1 = 8$ and TA = 1024.

Table VIII-2 contains the minimum and maximum times in milliseconds for Job 2. The three double columns of data are the times for an F/R which has 10, 20, and 40 (n_2) attributes. The first four rows of data are for $n_5 = 4$, the next four rows of data are for $n_5 = 8$, and the last four rows of data are for $n_5 = 16$. The first two rows of data in each group of four provide the times for the sequential computer with Type I and Type II synonyms, respectively. The second two rows of data in each group of four provide the times for the DDP with Type I and Type II synonyms, respectively. The parameters and their values for the minimum columns of data are $P = P_1 = n_1 = n_3 = n_4 = 1$, TA = 256, and $n_6 = 0$. The parameters and their values for the maximum columns of data are $P = P_1 = n_1 = n_4 = n_6 = 8$, $n_3 = 10$, and TA = 1024. The first double column of data for the last four rows are blank because n_5 is greater than n_2 which is an infeasible situation.

Table VIII-3 contains the minimum and maximum times in milliseconds for Jobs 3 and 4. The two columns of data are the times

Job 1

		$n_2 = 10$		$n_2 = 20$		$n_2 = 40$	
		min	max	min	max	min	max
Seq	Type I	0.9	2.0	1.7	3.5	3.3	6.5
	Type II	0.8	1.7	1.6	3.2	3.2	6.2
DDP	Type I	0.1	0.4	0.3	0.7	0.6	1.2
	Type II	0.2	0.4	0.3	0.6	0.6	1.1

(Time in milliseconds)

Table VIII-1. Job 1 Resultant Times for the Sequential (Seq) Computer and the Dictionary/Directory Processor (DDP).

Job 2

			$n_2 = 10$		$n_2 = 20$		$n_2 = 40$	
			min	max	min	max	min	max
$n_5 = 4$	Seq	Type I	1.6	7.8	2.4	9.3	4.0	12.0
		Type II	0.7	5.9	0.7	5.9	0.7	5.9
	DDP	Type I	0.2	0.8	0.2	0.8	0.2	0.8
		Type II	0.2	0.8	0.2	0.8	0.2	0.8
$n_5 = 8$	Seq	Type I	2.3	13.7	3.1	15.2	4.7	17.8
		Type II	1.4	11.7	1.4	11.7	1.4	11.7
	DDP	Type I	0.3	1.2	0.3	1.2	0.3	1.2
		Type II	0.2	1.0	0.2	1.0	0.2	1.0
$n_5 = 16$	Seq	Type I			4.5	26.9	6 1	29.5
		Type II			2.8	23.5	2.8	23.5
	DDP	Type I			0.5	1.8	0.5	1.8
		Type II			0.4	1.8	0.4	1.8

(Time in milliseconds)

Table VIII-2. Job 2 Resultant Times for the Sequential (Seq) Computer and the Dictionary/Directory Processor (DDP).

Jobs 3 and 4

			$n_3 = 1$ min	$n_3 = 10$ max
$n_6 = 1$	Seq	Job 3	0.2	2.6
		Job 4	0.2	2.7
	DDP	Job 3	0.1	0.3
		Job 4	0.1	0.3
$n_6 = 4$	Seq	Job 3	0.8	6.8
		Job 4	0.2	2.8
	DDP	Job 3	0.3	1.2
		Job 4	0.1	0.3
$n_6 = 8$	Seq	Job 3	2.0	12.5
		Job 4	0.2	2.8
	DDP	Job 3	0.7	2.3
		Job 4	0.1	0.3

(Time in milliseconds)

Table VIII-3. Jobs 3 and 4 Resultant Times for the Sequential (Seq) Computer and the Dictionary/Directory Processor (DDP).

for modifying an attribute which has 1 and 10 (n_3) associated F/R(s). The first four rows of data are for $n_6 = 1$, the next four rows of data are for $n_6 = 4$ and the last four rows of data are for $n_6 = 8$. The first two rows of data in each group of four provide the times for the sequential computer for Job 3 and Job 4, respectively. The second two rows of data in each group of four provide the times for the DDP for Job 3 and Job 4 respectively. The parameters and their values for the minimum column of data are $P = P_1 = 1$ and TA = 256. The parameters and their values for the maximum column of data are $P = P_1 = 8$ and TA = 1024.

A close evaluation of the above three tables yields the following results. The minimum ratio of times for the sequential computer to the DDP is 2 and occurs for Jobs 3 and 4 for $n_3 = 1$. The maximum ratio of times for the sequential computer to the DDP is 20 and occurs for Job 2 for Type I synonyms with $n_2 = 40$ and $n_5 = 4$. This range, 2 to 20, for the ratio in time between the sequential computer and the DDP provides one quantitative resultant measure for the DDP.

More generic results pertaining to the DDP and a sequential computer are obtainable by summarizing, by jobs, the conclusions that can be drawn from Figs. VII-8 through VII-13. For Job 1, the type of synonyms in the data base (Type I and Type II) only slightly affect the timings and the DDP is faster than the sequential computer. As the number of attributes per F/R increases, the ratio of the sequential computer to the DDP time also increases. For Job 2, the DDP is always faster than the sequential computer when the number of attributes per job (n_5)

is greater than six. Their lower bounds are approximately equal when n_5 is equal to one. The ratio of the sequential computer time to the DDP time increases as n_5 increases. For Job 3, the lower bound timing curve for the DDP indicates that the DDP is faster than a sequential computer implementation. The ratio of the sequential computer time to the DDP time increases as the number of synonyms per attribute increases. Job 3 performed after Job 2 is faster than if performed after Job 1. For Job 4, the lower bound timing curve for the DDP indicates that the DDP is faster than a sequential computer implementation. The ratio of the sequential computer time to the DDP time increases as the number of associated F/Rs of an attribute increase. Job 4 performed after Job 2 is in general faster than if performed after Job 1.

FUTURE RESEARCH

The mathematical base developed herein, sets and DP sets, should be pursued further. It is felt that this mathematical base should be investigated to determine how well it can model and therefore communicate non-numerical data processing functions. It appears reasonable to conclude that this base is appropriate for describing parsing and syntactical functions, which appear in many large software programs. The Push and Pull functions defined for DP sets provide an obvious method for modeling push-down and pop-up stacks plus those functions that operate on these stacks. This mathematical base may provide a complete

language that can be used for all levels of communication and data processing from software design and evaluation to implementation. A language with these capabilities is needed to assist in the creation of more correct and reliable software.

An extension to the above research area involves designing computer architectures for some of the non-numerical data levels and their respective functions. This research would be similar to the design and evaluation of the DDP. The effort could consider those problems existing in the Reserved Word level of data. As an example, it seems feasible that a computer architecture for those data involved with parsing and syntactical functions would be beneficial for DBM and for large software programs in general. Other areas of data processing should also be considered. For instance, Landson and Sargent [26] have designed and evaluated different architectures for priority queues. Two of the priority queues considered were First In First Out (FIFO) and Last In First Out (LIFO). These priority queues are similar to implementations and functions performed on pop-up and push-down stacks. This area of research should be pursued in order to increase a computer's operating efficiency and possibly reduce some of the costs of computing.

The final and most important area related to DBM is the design of an architecture for a "fully" relational DBMS. The term "fully" relational signifies that a user may specify his/her query by only stating the attribute names and values of interest. Or, stated differently, the user does not have to know which File

or Relationship (F/R) each attribute in the query is related to. The DBMS should determine if the query can be fulfilled before processing. All the basic architecture is logically contained in the proposed DDP. Required is an efficient algorithm that can locate the one or more F/Rs that can fulfill the query by using AM2, ARRAY III, AM3, and AM4. The algorithm must first determine that all the attributes are related and then create the most efficient way of relating them through their respective F/Rs.

Appendix A

DP sets can be used to model data structures such as Relational, Tree, Network, and the Entity Set Model. Consider the following relation S where the domains are S#, SNAME, STATUS, and CITY (See Fig. V-1). The Relation can be modeled as

$$S = \{A_1 \cup \{\not\emptyset^3\} \cup P_s^3(A_2) \cup \{\not\emptyset^9\} \cup P_s^9(A_3) \cup \{\not\emptyset^{16}\} \cup P_s^{16}(A_4)\}$$

where

$A_1 = \{S,\#\}$,

$A_2 = \{S,N,A,M,E\}$,

$A_3 = \{S,T,A,T,U,S\}$, and

$A_4 = \{C,I,T,Y\}$ are Normal DP sets.

The same procedure can be used to model an occurrence of S by replacing A_i by its respective j-th occurrence ($A_{i,j}$).

A tree structure can be depicted by a diagram similar to the figure shown in Fig. A-1. If we assume that each node contains some finite number of attributes, then a high level DP set model of this structure can be constructed using sets and DP sets as follows:

$$\{\{B, P_1, C, P_2, D, P_3, /, A_1, A_2, \ldots, A_n\},$$
$$\{E, P_4, /, B_1, B_2, \ldots, B_k\},$$

$$\{/, C_1, C_2, \ldots, C_m\},$$

$$\{/, D_1, D_2, \ldots, D_S\}, \{/, E_1, E_2, \ldots, E_P\}\}$$

where a slash (/) is used as a delimiter, P_i is a DP set model of a pointer to the node previous to itself, and the other alpha characters are the DP set models of the node's and attribute's names. The alpha characters that are right of the slash are the DP set models for attribute names. Those to the left of the slash are the DP set models for node names.

A network is modeled in a similar way. Consider a model of a network shown in Fig. A-2 where each node contains some finite number of attributes. A very high level model of this network can be constructed as:

$$\{\{B, P_1, C, P_2, /, A_1, A_2, \ldots, A_n, /, B, P_3\}, \{A, P_3, C, P_4, /,$$

$$B_1, B_2, \ldots, B_k, /, A, P_1\}, \{/, C_1, C_2, \ldots, C_m, /,$$

$$A, P_2, B, P_4\}\},$$

where the addition to the tree DP set model is the names and pointers following the second delimiter which points to the node below itself.

The Entity set model's basic building block, i.e. Entity Name Set Name/Role Name/Entity Name or its basic triplet can be modeled as a DP set. Consider the triplet modeled as a DP set where the first element is the Entity Name Set Name (E.N.S.N.), the second element is the Role Name (R.N.), and the third element is an actual occurrence of the data described by the first two

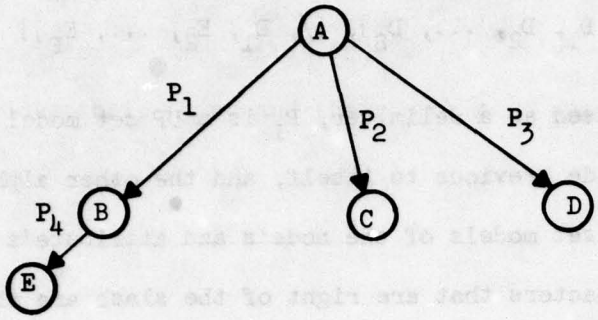


Figure A-1. Tree Structure.

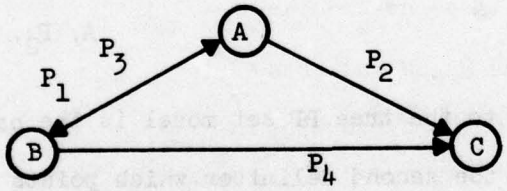


Figure A-2. Network Structure.

elements. The following is the format of the basic triplet modeled by a Normal DP set:

{{E,N,S,N}, {R,N}, {O,C,C,U,R,R,E,N,C,E}}.

An actual occurrence of a triplet might be:

{{P,A,R,T}, {P,A,R,T,$\not\emptyset$,S,U,P,P,L,I,E,D}, {P,A,R,T}}.

Appendix B

The two major techniques investigated for performing an equal to search (ETS) of a data dictionary and a data directory were hashing and explicit binary search (EBS). The hashing and hardware equations were developed in Chapter VII. This Appendix presents the equations for an EBS technique and the comparison of these equations with the hardware and hashing. The derivation of the equations for the EBS technique is presented first with a comparison of results with the other techniques. Then the variance in time associated with the EBS is developed followed by a trade-off analysis of the hashing and EBS techniques.

For the EBS technique, a modified version of algorithm T, by Knuth [25], was utilized. The N items are assumed to be stored and maintained as a full binary tree and the other assumption made is that each item has equal probability of being searched. An item is stored in a "node" in the tree which is made up of two computer words. The first word contains the left link (LLINK), which points to the address of the first word in the "node" for the next smallest item, and the right link (RLINK), which points to the address of the first word in the "node" for the next larger item. The second word in a "node" contains the item. The last assumption made is that every item searched is within the binary tree and will be found.

The required (average) time for the MIX computer is determined by the following set of steps and their respective number of times of execution.

Statement (MIXAL)

Line	Location	Operation	Address	NTS	Explanation
01	LLINK	EQU	2:3		LLINK set = to the 2nd and 3rd Bytes
02	RLINK	EQU	4:5		RLINK set = to the 4th and 5th Bytes
03		LDA	K	1(2)	Load the item value to be searched into register A
04		LD2	ROOT	1(2)	Load register 2 with the address of the root node
05		JMP	2F	1(1)	Jump to Line 08
06	4H	LD2	0,2(RLINK)	C2(2)	Load register 2 with the contents of the 4th and 5th Bytes of the word addressed in the current contents of register 2.
07		J2Z	EXIT	C2(1)	Exit if register 2 is = 0
08	2F	COMPA	1,2	C(2)	Compare register A with the word located in the address = (1+ contents of register 2)

Line	Location	Operation	Address	NTS	Explanation
09		JG	4H	C(1)	Jump to line (6) if the comparison in line 08 is "greater"
10		JE	SUCCESS	C1(1)	Jump to line (?) or a statement labeled "success" meaning the proper node was found
11		LD2	0,2(LLINK)(C1-S)(2)		Load register 2 with the contents of the 2nd and 3rd Bytes of the word addressed in the current contents of register 2
12		J2NZ	2F	(C1-S)(1)	Jump to line 08 if register 2 is $\neq 0$.

The time for the execution of this algorithm is calculated by summing up the number of times each statement is executed multiplied by U, where U is a unit of time or a relative measure; described by Knuth [24] such that

> . . .ADD, SUB, All Load operations, all STORE operations (including STZ), all shift commands and all comparison operations take two units of time. MOVE requires one unit plus two for each word moved. MUL requires 10 and DIV requires 12 units. Execution time for floating-point operations is unspecified. All remaining operations take one unit of time, plus the time the computer may idle on the IN, OUT, IOC, or HLT instructions.

Given this definition of U and the following parameter definitions, a function for an average response time can be developed. The

parameters are:

1) C, average number of comparisons made;
2) C1, average number of times the searching item value is $\leq$ "node"'s item value;
3) S, S = 1 if the search is successful, 0 otherwise; and
4) C2, average number of times the searching item value > "node"'s item value ($\therefore$ C = C1 + C2).

Then, on the average (since it was assumed the tree was a complete binary tree), C1 - S $\approx$ C2 and C1 $\approx$ (C+S)/2. Using this information, an average response time ($\overline{R.T.}$) can be computed as $(6.5C - 2.5S + 5)U = \overline{R.T.}$; where from [25] C, the average number of comparisons in a successful search can be represented as:

$$C = \lfloor \log_2 N \rfloor + 1 - (2^{\lfloor \log_2 N+1 \rfloor} - \lfloor \log_2 N \rfloor - 2)/N,$$

where $\lfloor X \rfloor$ = greatest integer $\leq$ X.

These equations can be used to calculate values of $\overline{R.T.}$ and the response time (R.T.) for different values of N and U. The results of some sample calculations are shown in Table B-1.

These models are fine except in DBM problems, attribute, and relationship or file names for example, are usually much larger than 31 bits. Therefore, models must be developed for the number of words per item (w) greater than one. For the Dictionary/ Directory Processor (DDP) the response time is

R.T. = $(w(7.38) + 0.13)\mu$seconds and for the sequential machine,

DDP Response Time	Hashing Response Time (R.T.)	EBS Average Response Time ($\overline{R.T.}$)	No. of Nodes (N)	U (Time Unit)
7.2μsec.	.34μsec.	48.0μsec.	256	1μsec.
7.2μsec.	34μsec.	61.0μsec.	1024	1μsec.
7.2μsec.	34μsec.	67.5μsec.	2048	1μsec.
7.2μsec.	68μsec.	96.0μsec.	256	2μsec.
7.2μsec.	68μsec.	122.0μsec.	1024	2μsec.
7.2μsec.	68μsec.	135.0μsec.	2048	2μsec.

Table B-1. Search Times of the Dictionary/Directory Processor (DDP), a Hashing Technique and an Explicit Binary Search Technique for One Computer Word Long Keys.

$$R.T. = (40 + (w-2)21)\ U$$

where $2 \leq w \leq 8$.

The upper bound of 8 exists because of the STARAN word length which is only 256 bits. For the MIX computer

$$\overline{R.T.} = (1 + 8C + 6(w-1) + \sum_{j=2}^{w} 7(C-1)r_j)\ U$$

where r_j represents the average fractional number of times the j-th word in a "node" will need to be compared and that the "node" is not equivalent to the item being searched. This equation is based on the assumption that each node requires the same number of fields or words to store each item value (i.e. fixed length items). The results of some sample calculations with various values of N for two different conditions are shown in Table B-2.

In Table B-2 for the EBS technique w and r_j were varied, but inherent in the times provided so far is the fact that the number of comparisons (C) made is an average of a random variable whose values represent the number of comparisons that have to be made until the item is found. The previous expression for the average number of comparisons (C) was obtained from Knuth [25]. E(C), the expected number of comparisons can also be obtained. These expected or average number of comparisons can be misleading if considered without knowing an estimate of the variance associated with them.

To determine the variance, consider the following derivation. Let L be a random variable such that its values represent the level of nodes in a full binary tree, then L = 0 represents the

TIME IN μSEC.

		w = 2		w = 3		w = 4		w = 5	
		I	II	I	II	I	II	I	II
EBS	N = 256; C ≈ 7;	84	105	100.5	153	111.75	201	120.38	249
	N = 1024; C ≈ 9;	107	135	127	197	140	259	149.5	321
	N = 2048; C ≈ 10;	118.5	150	140.3	219	154.1	288	164.1	357
	N = 8192; C ≈ 12;	141.5	180	166.8	263	182.4	346	193.2	429
HARDWARE		14.89		22.27		29.65		37.03	
HASHING		40.00		61.00		82.00		103.00	

For the MIX machine U = 1μsec. and I assumes $r_2 = \frac{1}{2}$, $r_3 = \frac{1}{4}$, $r_4 = \frac{1}{8}$, $r_5 = \frac{1}{16}$, $r_6 = \frac{1}{32}$, $r_7 = \frac{1}{64}$, and $r_8 = \frac{1}{128}$.

II assumes $r_2 = r_3 = r_4 = r_5 = r_6 = r_7 = r_8 = 1$

Table B-2. Search Time Comparisons for Variable Word Size Keys and Nodes.

		w = 6		w = 7		w = 8	
		I	II	I	II	I	II
EBS	N = 256; C ≈ 7;	127.69	297	134.34	345	140.67	393
	N = 1024; C ≈ 9;	157.25	383	164.13	445	170.56	507
	N = 2048; C ≈ 10;	172.0	426	179.0	495	185.5	564
	N = 8192; C ≈ 12;	201.6	512	208.8	595	215.4	678
HARDWARE		44.41		51.79		59.17	
HASHING		124.00		145.00		166.00	

For the MIX machine U = 1μsec., and I assumes $r_2 = \frac{1}{2}$, $r_3 = \frac{1}{4}$, $r_4 = \frac{1}{8}$, $r_5 = \frac{1}{16}$; $r_6 = \frac{1}{32}$

$r_7 = \frac{1}{64}$, and $r_8 = \frac{1}{128}$.

II assumes $r_2 = r_3 = r_4 = r_5 = r_6 = r_7 = r_8 = 1$

Table B-2. Search Time Comparisons for Variable Word Size Keys and Nodes.

0 level, L = 1 represents the first level, etc. (See Fig. B-1). Since it was assumed earlier that the probability (P) of accessing any node in the binary tree is a constant, then for N nodes the $P(L = 0) = 1/N$, $P(L = 1) = 2/N$, ..., $P(L = i) = 2^i/N$. To find the different moments of the random variable L, its probability generating function, G(Z), can be utilized,

$$G(Z) = \sum_{\forall i} Z^i(P(L = i)).$$

The expected value of L, E(L), is determined by differentiating G(Z) and solving for (d(G(Z))/dZ) with Z = 1. This yields

$$\left.\frac{d(G(Z))}{dZ}\right|_{Z=1} = \left.\sum_{\forall i} iZ^{i-1}(P(L=i))\right|_{Z=1} = \sum_{\forall i} i(P(L=i)) = E(L) = \sum_{\forall i} i(2^i/N).$$

The second factorial moment of L is:

$$\left.\frac{d^2(G(Z))}{dZ^2}\right|_{Z=1} = \left.\sum_{\forall i} i(i-1)\, Z^{i-2}\left(\frac{2^i}{N}\right)\right|_{Z=1} = \sum_{\forall i} i(i-1)\left(\frac{2^i}{N}\right) = E[L(L-1)].$$

From the above two moments (i.e. E(L) and E[L(L-1)]) the variance of L, $\sigma^2(L)$, can be obtained:

$$\sigma^2(L) = E[L(L-1)] + E(L) - (E(L))^2,$$

$$= \sum_{\forall i} i(i-1)\left(\frac{2^i}{N}\right) + \sum_{\forall i} i\left(\frac{2^i}{N}\right) - \left(\sum_{\forall i} i\left(\frac{2^i}{N}\right)\right)^2.$$

If, however, the tree is not completely balanced, say for N = 256, then the probability density function for L is defined as:

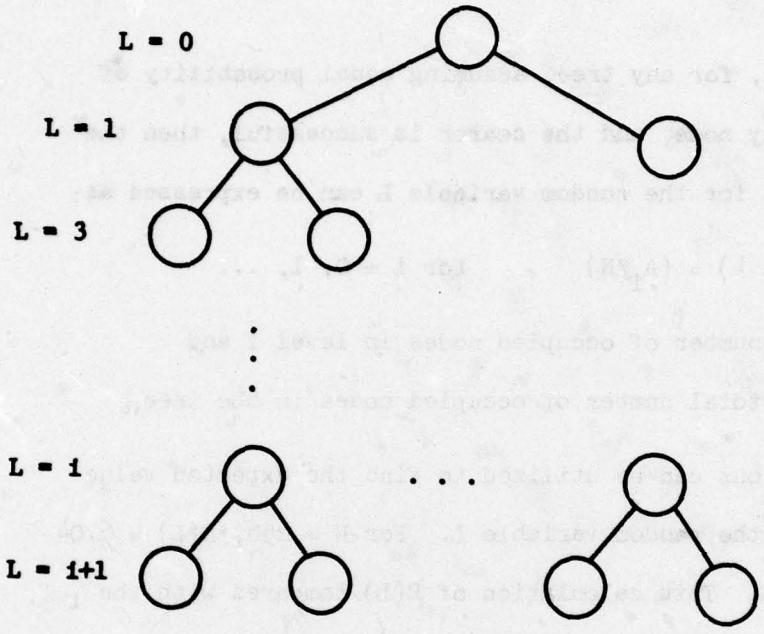


Figure B-1. Binary Tree.

$$P(L=i) = \begin{cases} 2^i/N & \text{for } i = 0, 1, \ldots, 7 \\ 1/N & \text{for } i = 8 \\ 0 & \text{otherwise} \end{cases}$$

where N = 256.

In general, for any tree, assuming equal probability of searching for any node, and the search is successful, then the density function for the random variable L can be expressed as:

$$P(L = i) = (A_i/N) \qquad \text{for } i = 0, 1, \ldots$$

where A_i is the number of occupied nodes in level i and

N is the total number of occupied nodes in the tree.

The above equations can be utilized to find the expected value and variance of the random variable L. For N = 256, E(L) = 6.04 and $\sigma^2(L) = 1.75$. This calculation of E(L) compares with the statistical estimate of C where $E(L) + 1 \approx C$, i.e. $(6.04 + 1) \approx 7$. To get a feeling of the spread in time required to perform an ETS with N = 256 and for w = 1 and 2, consider the results shown in Table B-3. Comparing the times shown in B-3 and B-2, it seemed reasonable to use the hashing algorithm's times in evaluating the proposed hardware.

It has been shown by Knuth that hashing techniques are much faster than binary searching techniques as N get large. In different problem areas, the value of N might be small but the number of ETSs that have to be made might be very large. The trade-off analysis for w = 1 would be to find what value of N (a power of 2) will the EBS be faster than the hashing algorithm.

N	W	C = E(C) - σ	C = E(C)	C = E(C) + σ
		C = 5.68	C = 7	C = 8
256	1	(39.42)U	(48)U	(54.5)U
256	2	(68.82)U	(84)U	(95.5)U

where $C = 5.68 \approx (E(L) - \sigma(L) + 1)$,

$C = 7 \approx (E(L) + 1)$,

$C = 8$ (Maximum value of C) $< (E(L) + 1 + \sigma(L))$, and

$r_2 = 1/2$.

Table B-3. Variable Times of the Explicit Binary Technique for the Mean and ± one Standard Deviation of the Number of Compares.

This can be determined by setting the timing equation of the EBS technique equal to the time for the hashing technique and solving for N, i.e.

$$(6.5[\lfloor \log_2 N \rfloor + 1(2^{\lfloor \log_2 N+1 \rfloor} - \lfloor \log_2 N \rfloor - 2)/N] + 2.5)U = 34U$$

where $N \approx 64$.

This means, given all the above assumptions, that on the average for $w = 1$ and $N \leq 64$, an EBS technique will be faster than the hashing function which has no collisions.

Carrying this one step further to $w = 2$, the value of N reduces to 16 and solving for r_2 as follows yields:

$$(34 + 7(19/8)\ r_2)U = (40)U$$

where $r_2 \approx 0.36$.

This implies that if r_2 is $\leq .36$ and if $N \leq 16$, then on the average the EBS technique will be faster than the hashing technique presented. This same technique can be extended to $w = 3, 4, \ldots, 8$ to determine the values of the (r_j)'s such that the hashing and EBS techniques are on the average approximately equivalent.

References

[1] Anderson, George A. and Richard Y. Kain, "A Context-Addressed Memory Designed for Data Base Applications," Proceedings of the 1976 International Conference on Parallel Processing, 1976, pp. 191-195.

[2] Baum, Richard I. and David K. Hsiao, "A Data Secure Computer Architecture," COMPCON '76, 1976, pp. 113-117.

[3] Benjamin, R., "A Generational Perspective of Information Systems Development," Communications of the ACM, Vol. 15, No. 7, July 1972, pp. 640-643.

[4] Berra, P. Bruce, "Some Problems in Associative Processor Applications to Data Base Management," National Computer Conference, 1974, pp. 1-5.

[5] Berra, P. Bruce and Ashok K. Singhania, "A Multiple Associative Memory Organization for Pipelining a Directory to a Very Large Data Base," COMPCON '76, 1976, pp. 109-112.

[6] Bush, J. A., S. Y. W. Su, G. J. Lipovski, J. K. Watson, and S. J. Ackerman, "Some Implementations of Segment Sequential Functions," The 3rd Annual Symposium on Computer Architecture, 1976, pp. 178-185.

[7] Childs, David L., "Feasibility of a Set-Theoretic Data Structure: A General Structure Based on a Reconstituted Definition of a Relation," IFIP Congress 1968, North Holland, Amsterdam, 1968, pp. 420-430.

[8] CODASYL, "Data Base Task Group - April '71 Report," ACM, April, 1971.

[9] CODASYL, "A Feature Analysis of Generalized Data Base Management Systems," CACM, May, 1971.

[10] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, June, 1970, pp. 377-387.

[11] Datapro Research Corporation, A Buyer's Guide to Data Base Management Systems, Delran, New Jersey,: Datapro Research Corporation, 1974.

[12] Date, C. J., An Introduction to Database Systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1976.

[13] DeFiore, C. R. and P. B. Berra, "A Data Management System Utilizing an Associative Memory," AFIPS Conference Proceedings, Vol. 42, June, 1973, pp. 181-185.

[14] DeFiore, C. R. and P. B. Berra, "A Quantitative Analysis of the Utilizations of Associative Memories in Data Management," IEEE Transactions on Electronic Computers, February, 1974.

[15] DeMartinis, Manlio, G. Jack Lipovski, Stanley Y. W. Su, and J. K. Watson, "A Self Managing Secondary Memory System," The 3rd Annual Symposium on Computer Architecture, 1976, pp. 186-194.

[16] Foster, Caxton, C., Computer Architecture, Van Nostrand Reinhold Company, 1970.

[17] Fry, J. P., "Managing Data is the Key to MIS," Computer Decisions, January, 1971, pp. 6-10.

[18] Goodyear Aerospace Corporation, "STARAN Reference Manual," GER-15636B, September, 1974.

[19] Healy, D. Leonard, "A Character-Oriented Context-Addressed Segment-Sequential Storage," The 3rd Annual Symposium on Computer Architecture, 1976, pp. 172-177.

[20] Healy, L. D., G. J. Lipovski, and K. L. Doty, "The Architecture of a Context Addressed Segment-Sequential Storage," Proc. FJCC, 1972, pp. 691-701.

[21] Hollaar, Lee Allen, "A List Merging Processor for Inverted File Information Retrieval Systems," Report No. UIUCDCS-R-75-762, University of Illinois, 1975.

[22] Hsiao, D. and F. Harrary, "A Formal System for Information Retrieval from Files," Communications of the ACM, February, 1970, pp. 67-73.

[23] Joint Guide-Share Data Base Requirements Group, "Data Base Management Systems Requirements," November 11, 1970.

[24] Knuth, D., The Art of Computer Programming, Vol. I/Fundamental Algorithms, Addison-Wesley Publishing Company, Reading, Massachusetts, 1968.

[25] Knuth, D., The Art of Computer Programming, Vol. III/Sorting and Searching, Addison-Wesley Publishing Company, Reading, Massachusetts, 1972.

[26] Landson, B. M. and R. G. Sargent, "A Comparison of Sequential and Associate Computing of Priority Queues," SIGIR-SIGARCH-SIGMOD Third Workshop on Computer Architecture for Non-Numeric Processing, May 17-18, 1977, pp. 77-78.

[27] Lin, C. S., Diane C. P. Smith, and John M. Smith, "The Design of a Rotating Associative Memory for Relational Database Application," ACM Transactions on Database Systems, March, 1976, Vol. 1, No. 1, pp. 53-65.

[28] Linde, R. R., R. Gates, and T. Peng, "Associative Processor Applications to Real-Time Data Management," AFIPS Conference Proceedings, Vol. 42, June, 1973, pp. 187-195.

[29] Love, H. H., "An Efficient Associative Processor Using Bulk Storage," Proceedings of 1973 Sagamore Computer Conference on Parallel Processing, 1973, pp. 103-112.

[30] Minsky, N., "Rotating Storage Devices as Partially Associative Memories," Proceedings of 1972 FJCC, Vol. 41, Part 1, 1972, pp. 587-596.

[31] Moulder, R., "An Implementation of a Data Management System on an Associative Processor," AFIPS Conference Proceedings, Vol. 42, June, 1973, pp. 171-176.

[32] Ozkarahn, E. A., S. A. Schuster, and K. C. Smith, "A Data Base Processor," Technical Report CSRG-43, University of Toronto, November, 1974.

[33] Ozkarahn, E. A., S. A. Schuster, and K. C. Smith, "RAP - An Associative Processor for Data Base Management," AFIPS Conference Proceedings, 1975, pp. 379-387.

[34] Parker, J. L., "A Logic per Track Retrieval System," IFIP Congress, 1971, pp. TA-4-146 - TA-4-150.

[35] Parhami, B., "A Highly Parallel Computing System for Information Retrieval," Proc. FJCC, 1972, pp. 681-690.

[36] Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structures and Accessing in Data Base Systems," IBM Systems J 12, 1973, pp. 30-93.

[37] Singhania, Ashok K., "A Multiple Associative-Memory System for Pipelining a Directory to a Very Large Data Base," unpublished doctoral dissertation, Syracuse University, May, 1977.

[38] Su, Stanley Y. W., and G. J. Lipovski, "On Large Associative Memory Systems for Non-Numeric Processing," unpublished paper.

[39] Thurber, Kenneth J. and Leon D. Wald, "Associative and Parallel Processors," Computing Surveys, Vol. 7, No. 4, December 1975, pp. 215-255.

[40] Zenna, Peter W. and Robert L. Johnson, Elements of Set Theory, Allyn and Bacon, Inc., Boston, Massachusetts, 1964.